A MARSHALL CAVENDISH 37 COMPUTER COURSE IN WEEKLY PARTS

INPUT

LEARN PROGRAMMING - FOR FUN AND THE FUTURE

UK £1.00
Republic of Ireland £1.25
Malta 85c
Australia $2.25
New Zealand $2.95



INPUT

Vol. 3 **No 37**

MACHINE CODE 38

CLIFFHANGER: KEEPING THE SCORE 1145

Add a scoring routine to Cliffhanger. Keep track of how well Willie's doing

GAMES PROGRAMMING 38

FOX AND GEESE GAME: 3 1152

Complete the intelligent game with routines to control the fox and the geese, and look ahead

BASIC PROGRAMMING 77

PREDICTING THE UNPREDICTABLE 1158

Use statistical methods and computer modelling to predict happenings in the real world

BASIC PROGRAMMING 78

PATTERNS FROM NATURE 1164

Generate complex and beautiful patterns from simple beginnings

APPLICATIONS 24

TAILORING SPREADSHEETS 1172

Continue the spreadsheet program. Soon you'll be able to plan your expenses or business

INDEX

The last part of INPUT, Part 52, will contain a complete, cross-referenced index. For easy access to your growing collection, a cumulative index to the contents of each issue is contained on the inside back cover.

PICTURE CREDITS
Front cover, Digital Arts. Pages 1144, 1146, 1148, Alistair Graham. Pages 1152, 1154, 1155, Peter Beard. Pages 1159, 1160, Digital Arts. Page 1163, J D Audio Visual. Pages 1164, Spectrum/Paul Chave. Pages 1166, 1171, NASA/Paul Chave. Pages 1168, 1169, 1173, Peter Reilly. Pages 1172, 1173, 1174, 1175, 1176, Chen Ling.

Published by Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA, England. Typeset by MS Filmsetting Limited, Frome, Somerset. Printed by Cooper Clegg Web Offset Ltd, Gloucester and Howard Hunt Litho, London.



There are four binders each holding 13 issues.

HOW TO ORDER YOUR BINDERS

UK and Republic of Ireland:
Send £4.95 (inc p & p) (IR£5.95) for each binder to the address below:
Marshall Cavendish Services Ltd, Department 980, Newtown Road, Hove, Sussex BN3 7DN
Australia: See inserts for details, or write to INPUT, Times Consultants, PO Box 213, Alexandria, NSW 2015
New Zealand: See inserts for details, or write to INPUT, Gordon and Gotch (NZ) Ltd, PO Box 1595, Wellington
Malta: Binders are available from local newsagents.

BACK NUMBERS

Back numbers are supplied at the regular cover price (subject to availability).
UK and Republic of Ireland:
INPUT, Dept AN, Marshall Cavendish Services, Newtown Road, Hove BN3 7DN
Australia, New Zealand and Malta:
Back numbers are available through your local newsagent.

COPIES BY POST

Our Subscription Department can supply copies to any UK address regularly at £1.00 each. For example the cost of 26 issues is £26.00; for any other quantity simply multiply the number of issues required by £1.00. Send your order, with payment to:
Subscription Department, Marshall Cavendish Services Ltd, Newtown Road, Hove, Sussex BN3 7DN
Please state the title of the publication and the part from which you wish to start.

HOW TO PAY: Readers in UK and Republic of Ireland: All cheques or postal orders for binders, back numbers and copies by post should be made payable to:
Marshall Cavendish Partworks Ltd.

QUERIES: When writing in, please give the make and model of your computer, as well as the Part No., page and line where the program is rejected or where it does not work. We can only answer specific queries – and please do not telephone. Send your queries to INPUT Queries, Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA.

INPUT IS SPECIALLY DESIGNED FOR:

The SINCLAIR ZX SPECTRUM (16K, 48K, 128 and +), COMMODORE 64 and 128, ACORN ELECTRON, BBC B and B+, and the DRAGON 32 and 64.

In addition, many of the programs and explanations are also suitable for the SINCLAIR ZX81, COMMODORE VIC 20, and TANDY COLOUR COMPUTER in 32K with extended BASIC. Programs and text which are specifically for particular machines are indicated by the following symbols:

SPECTRUM 16K, 48K, 128, and +
COMMODORE 64 and 128
ACORN ELECTRON, BBC B and B+
DRAGON 32 and 64
ZX81
VIC 20
TANDY TRS80 COLOUR COMPUTER

CLIFFHANGER: KEEPING THE SCORE

The 'CLIFFHANGER' listings published in this magazine and subsequent parts bear absolutely no resemblance to, and are in no way associated with, the computer game called 'CLIFF HANGER' released for the Commodore 64 and published by New Generation Software Limited.

No game is complete without a scoring routine, and even Willie won't be satisfied with just the bits of his picnic. Enter these routines and play the numbers game

The scoring routine must also scroll on the appropriate screen and print up the score, level and lives on the screen.

The following program calls up the appropriate screen, prints up the score and number of lives Willie has left and plays the tune:

```
org 58676
call lsi
call scp
ld hl,119
ld a,(57343)
ld b,48
add a,b
call asc
ld a,41
call print
call tune
ret
        org 58303
lsi     *
        org 58174
asc     *
        org 58217
print   *
        org 60000
tune    *
```

ON THE STARTING BLOCKS

The call lsi calls the routine that scrolls on the screen. This takes care of which screen is required, through its elb routine.

Then the scp routine is called. This is the one that actually prints up the score and has not been given to you yet, so don't run the program until you have keyed in the scp program given below. HL is loaded with 119. This is print position of the number of lives. The print routine is going to be called and the HL register is used to carry the print position into it.

A is loaded with the contents of 57,343. This is the memory location set aside to hold the number of lives that Willie's got left. The number 48 is loaded into B and the contents of the two registers are added together to give the ASCII code for the number that needs to be printed.

The asc routine is then called. This is the routine which returns the pointer to the image data for the numeral needed in ROM. A is then loaded with 41 to set the colour of the letter and the print routine is called. This actually prints the number of lives Willie has left. The tune routine is then called which plays the tune. Note that the music routine on page 966 must now be re-assembled with an origin of 60000—otherwise it will be overwritten by the routine that follows it.



SCORING

This is the scp routine called in the routine above that actually prints the score on the screen:

```
       org 58939
scp    ld hl,55
       ld ix,57337
       ld b,6
scq    push bc
       ld a,(ix+0)
       ld b,48
       add a,b
       call asc
       ld a,41
       call print
       inc hl
       inc ix
       pop bc
       djnz scq
       ret
       org 58174
asc    *
       org 58217
print  *
```

FIGURING THE FIGURES

HL is loaded with 55, the screen position the leading digit of the score will be printed at. The IX index register is loaded with 57,338, the address of the first byte which contains the score. The register can then be used as a pointer.

B is loaded with 5, the number of digits in the score. This counter is saved by pushing it onto the stack.

A is then loaded with the contents of the memory location pointed to by the IX register—in other words, the first byte containing the score. 48 is added to it to give the ASCII code and the asc routine is called to get the image data pointer in ROM.

Then A is loaded with 41 to set the colour and print is called to print out the digit.

HL is incremented to move along to the next print position—you'll notice that the score is being printed from the high digit to the low. IX is incremented to advance the score pointer to the next location containing a score digit. The score is stored as decimal digits—one in each memory byte—from the high digit to the low one, up memory.

The B counter is popped back off the stack and decremented by the djnz instruction. And if the counter has not been decremented to zero the processor jumps back to print up the next digit.

When it has been round the loop five times, printing up the five digits of the score, the processor drops out of the loop and returns.

On the Commodore it is necessary to have a routine at this stage which works out where positions are on the screen. In previous routines you have specified X and Y coordinates. From those coordinates, the appropriate screen memory location must be calculated.

The formula to do this is simple. There are 40 character squares across the screen, so you must multiply the Y coordinates by 40 and add the X coordinate to it, then add on 1024, which is the base address of the screen memory, to give the appropriate location.

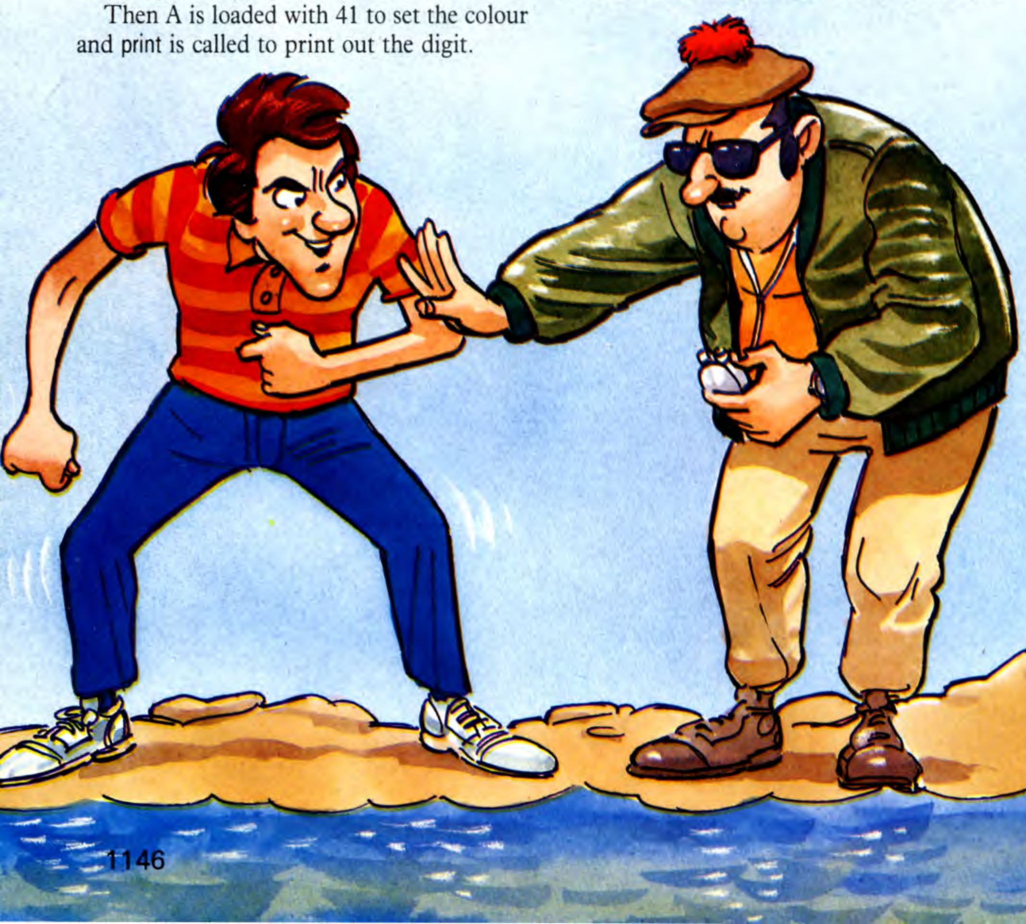
Memory location $Ø352 is used to store the X coordinate and $Ø353 is used to store the Y coordinate. And locations $Ø384 and $Ø385 are going to be used to store the resulting screen address. These locations are in the cassette buffer, but the cassette player is not going to be used while the game is being played and they are convenient RAM locations to use to pass parameters in and out of this multiplication routine.

```
      ORG 2Ø48Ø
      LDA $Ø353
      STA $Ø384
      LDA #Ø
      STA $Ø385
      CLC
      LDX #5
LOOP  CLC
      ROL $Ø384
      ROL $Ø385
      DEX
      BNE LOOP
      LDA $Ø353
      STA $Ø386
      LDA #Ø
      STA $Ø387
      LDX #3
LOOPJ CLC
      ROL $Ø386
      ROL $Ø387
      DEX
      BNE LOOPJ
      CLC
      LDA $Ø384
      ADC $Ø386
      STA $Ø384
      LDA $Ø385
      ADC $Ø387
      CLC
      ADC #4
      STA $Ø385
      CLC
      LDA $Ø352
      ADC $Ø384
      STA $Ø384
      LDA $Ø385
      ADC #Ø
      STA $Ø385
      LDA $Ø384
      STA $FB
      LDA $Ø385
      STA $FC
      LDY #Ø
      RTS
```

MULTIPLICATION

The Y coordinate in $Ø353 is loaded into the accumulator and stored directly in the low byte of the result, $Ø384. Then the high byte, $Ø385, is cleared by storing Ø in it.

Multiplication on an eight-bit processor is not an easy matter. It has to be done by a combination of shifts, rotates or repeated additions. To multiply a number by 40 it is easiest to multiply it by 8—that is, 2↑3—then multiply it by 32—which is 2↑5—and add the results together (32 + 8 = 40). Multiplication by powers of 2 is a relatively simple matter. To multiply by 2, you simply have to shift all the bits in a register one place to the left. So to multiply by 8 you have to shift the bits to the left three places—or shift them to the left one



place three times—and to multiply by 32 you have to shift them five places to the left, or one place five times.

The problem is dealing with the bits that are shifted out of the register.

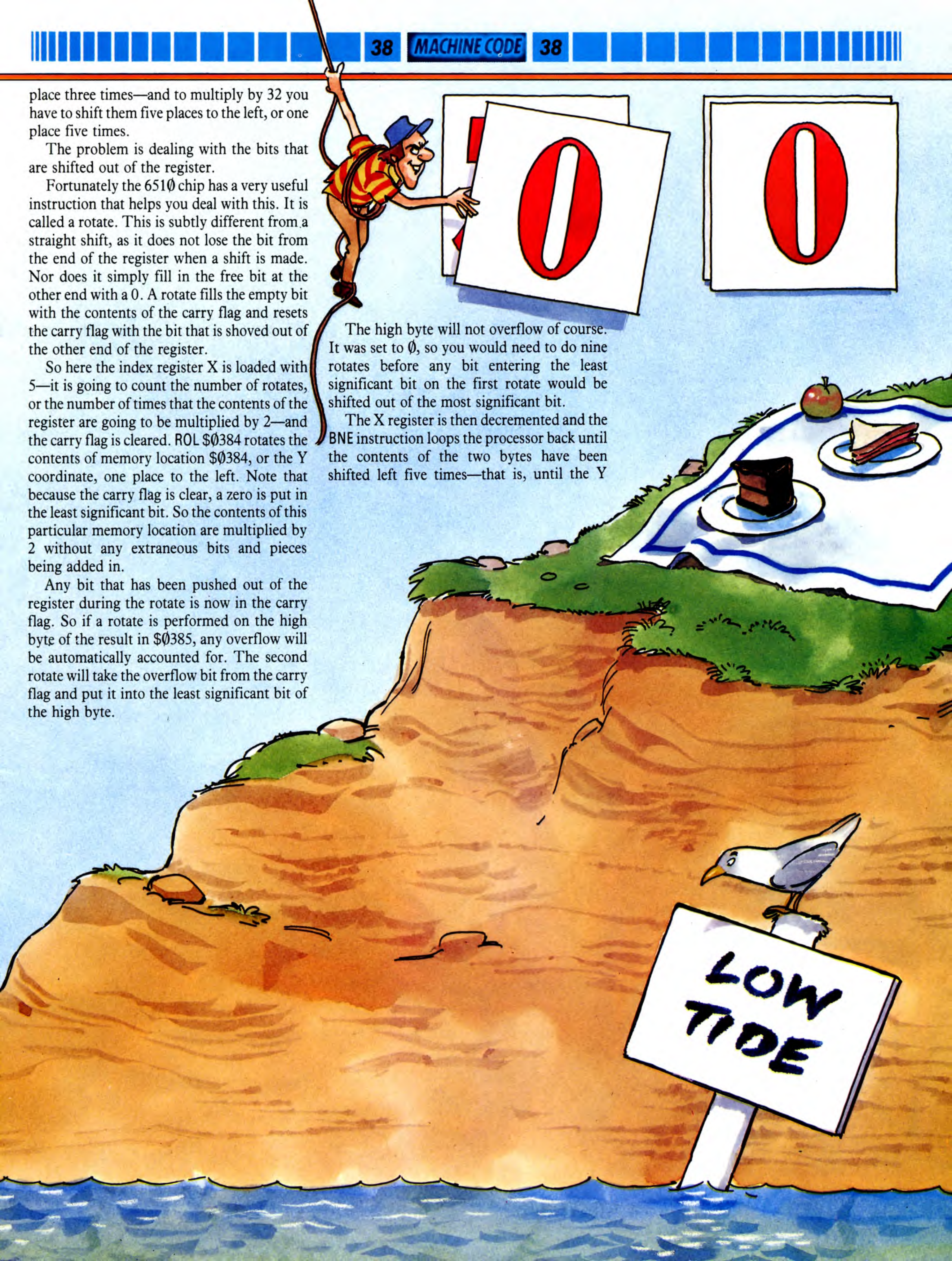
Fortunately the 651Ø chip has a very useful instruction that helps you deal with this. It is called a rotate. This is subtly different from a straight shift, as it does not lose the bit from the end of the register when a shift is made. Nor does it simply fill in the free bit at the other end with a 0. A rotate fills the empty bit with the contents of the carry flag and resets the carry flag with the bit that is shoved out of the other end of the register.

So here the index register X is loaded with 5—it is going to count the number of rotates, or the number of times that the contents of the register are going to be multiplied by 2—and the carry flag is cleared. ROL $Ø384 rotates the contents of memory location $Ø384, or the Y coordinate, one place to the left. Note that because the carry flag is clear, a zero is put in the least significant bit. So the contents of this particular memory location are multiplied by 2 without any extraneous bits and pieces being added in.

Any bit that has been pushed out of the register during the rotate is now in the carry flag. So if a rotate is performed on the high byte of the result in $Ø385, any overflow will be automatically accounted for. The second rotate will take the overflow bit from the carry flag and put it into the least significant bit of the high byte.

The high byte will not overflow of course. It was set to Ø, so you would need to do nine rotates before any bit entering the least significant bit on the first rotate would be shifted out of the most significant bit.

The X register is then decremented and the BNE instruction loops the processor back until the contents of the two bytes have been shifted left five times—that is, until the Y



coordinate has been multiplied by 32.

The Y coordinate in $0353 is then loaded into a temporary storage location at $0386 and the next byte, $0387, is set to 0. The contents of these two locations are going to be shifted left in exactly the same way as before, only this time the loop—and the shift—is only three times. So $0386 and $0387 end up containing the Y coordinate times 8.

Now the two results have to be added together. So the carry flag is cleared, the contents of $0384 are loaded into the accumulator and the contents of $0386 are added to them. The result is stored back in $0384, which is where it is needed.

Any overflow from this addition will now be in the carry flag, so when $0385 is loaded into the accumulator and added to the contents of $0386, any carry is automatically taken into account.

A further 4 is added into the high byte—$4 \times 256 = 1024$. This adds in the base address of the screen memory.

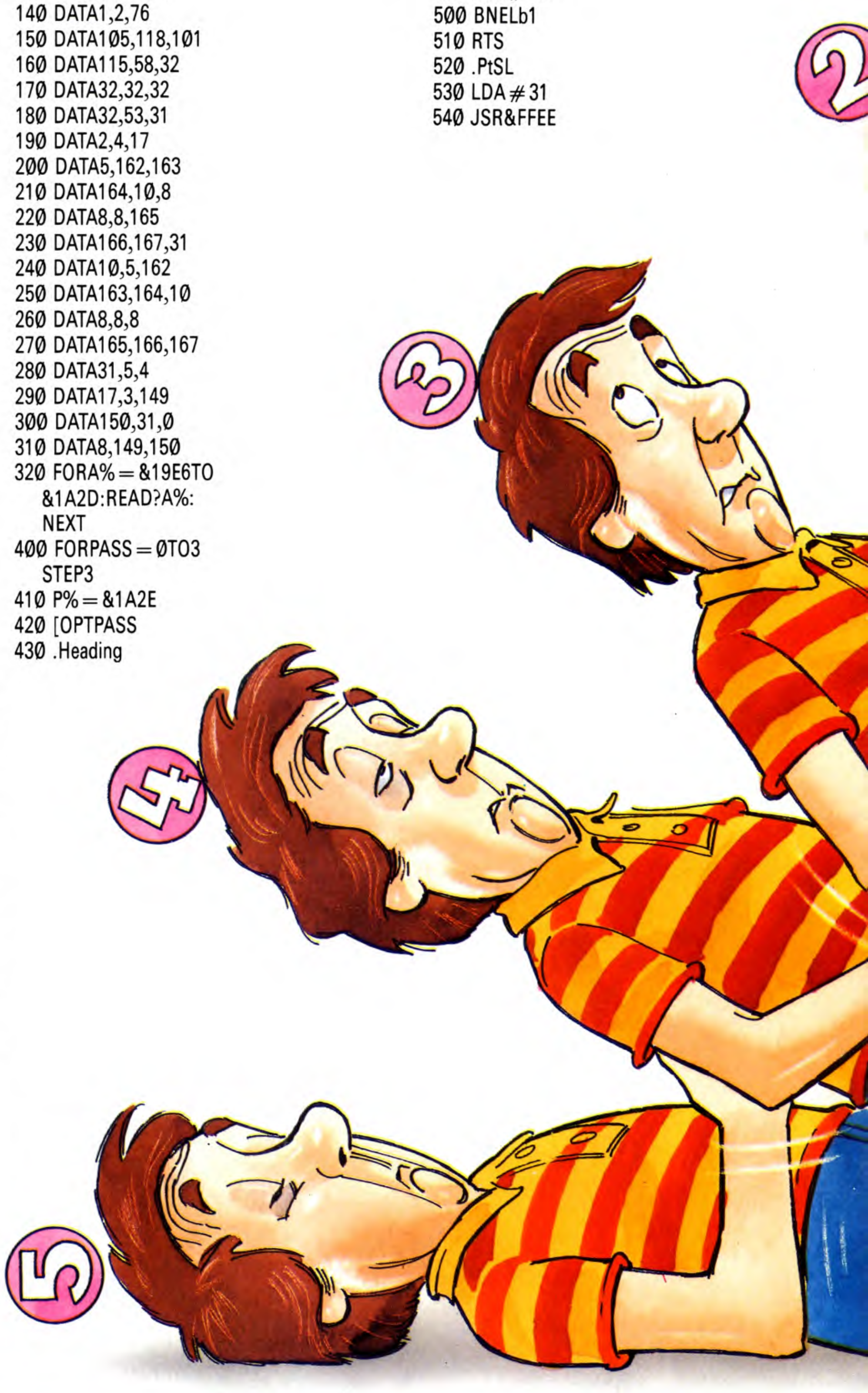
Then the X coordinate in $0352 is loaded into the accumulator and added to the low byte of the result in $0384. The result is stored back in $0384. Any overflow from this addition is left in the carry flag.

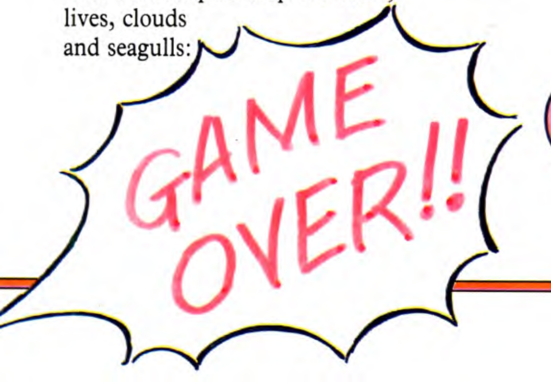
The high byte of the result is then loaded into the accumulator and has 0 added to it. Note, this is an add with carry. So if there is a carry from the addition of the low byte with the X coordinate it will be added in now. If not, the contents of the accumulator remain unchanged. Either way, they are stored back in $0385. Note that the contents of the high byte can't overflow. For that to happen, the resulting address would have to be more than 65,535 which the highest address permissible on the 64K Commodore.

The calculation is now complete, but for added convenience the result is copied from $0352 and $0353 into the zero-page memory locations $FB and $FC. Later, when the result needs to be accessed, say by indirect indexed address, the instructions can be shaved by a byte because zero page addressing is used and speed might be of the essence at that point in the game.

This routine prints up the score, lives, clouds and seagulls:

```
80 DATA31,1,1
90 DATA17,1,83
100 DATA99,111,114
110 DATA101,58,48
120 DATA48,48,48
130 DATA48,48,31
140 DATA1,2,76
150 DATA105,118,101
160 DATA115,58,32
170 DATA32,32,32
180 DATA32,53,31
190 DATA2,4,17
200 DATA5,162,163
210 DATA164,10,8
220 DATA8,8,165
230 DATA166,167,31
240 DATA10,5,162
250 DATA163,164,10
260 DATA8,8,8
270 DATA165,166,167
280 DATA31,5,4
290 DATA17,3,149
300 DATA150,31,0
310 DATA8,149,150
320 FORA%=&19E6TO
   &1A2D:READ?A%:
   NEXT
400 FORPASS=0TO3
   STEP3
410 P%=&1A2E
420 [OPTPASS
430 .Heading
440 LDX#0
450 .Lb1
460 LDA&19E6,X
470 JSR&1803
480 INX
490 CPX#72
500 BNELb1
510 RTS
520 .PtSL
530 LDA#31
540 JSR&FFEE
```







```
55Ø LDA#7
56Ø JSR&FFEE
57Ø LDA#1
58Ø JSR&FFEE
59Ø LDA#17
6ØØ JSR&FFEE
61Ø LDA#1
62Ø JSR&FFEE
63Ø LDX#Ø
64Ø .Lb2
65Ø LDA&8A,X
66Ø CLC
67Ø ADC#48
68Ø JSR&FFEE
69Ø INX
7ØØ CPX#6
71Ø BNELb2
72Ø LDA#31
73Ø JSR&FFEE
74Ø LDA#12
75Ø JSR&FFEE
76Ø LDA#2
77Ø JSR&FFEE
78Ø LDA&89
79Ø CLC
8ØØ ADC#48
81Ø JSR&FFEE
82Ø RTS
83Ø ]NEXT
```

HEADINGS

The DATA which gives the words 'score', 'lives', clouds and seagulls is read into a data table from &19E6 to &1A2D. And the machine code routine that uses it is assembled directly after it.

The assembly language program starts by initializing the index register X to Ø to act as a loop counter. The byte of data from the data table pointed to by the base address, offset by X, is loaded into the accumulator and printed at the top of the screen by jumping to the user-defined graphic subroutine which is located at &18Ø3.

X is then incremented and compared to 72 to check whether all the data has been output to the screen. If it hasn't, the processor branches back to deal with the next byte. If it has, the processor drops out of the loop, hits the RTS and returns.

FIGURING

Loading A with 31 and calling the routine at &FFEE, allows you to position the cursor. The 7 and 1 are the coordinates. Loading A with 17 and calling &FFEE sets the colour. Here colour 1 is used, but don't forget that you've redefined all the colours earlier.

X is initialized to Ø again and the number in zero-page memory location &8A, offset by X, is loaded into the accumulator. This

location and the five following it are used to store the score, in decimal digits, from the high digit to the low digit, up memory.

The number 48 is added to the figure to give the ASCII, which is output to the screen by calling &FFEE.

X is then incremented and the processor loops back to print out the next digit—unless it is the case that all six of them have been put up on the screen.

The cursor is then repositioned at 12, 2 and the number in &89—which is where the number of lives are stored—is converted into ASCII and output to the screen. The processor then returns again.

To test it with the other machine code in the machine CALL&1B32, putting the level of the screen in &83.

This routine scrolls on the appropriate screen and prints up the sun, the lives and the score—and makes sure that the numbers that are going to be displayed on the screen do not bump into each other:

```
        ORG 19489
        JSR $4AA5
        LDX #18Ø7
        LDU #17544
        LDB #5
SCPR    LDA #3
SCPRI   PULU Y
        STY ,X++
        DECA
        BNE SCPRI
        LEAX 26,X
        DECB
        BNE SCPR
        JSR PRSC
        LDX #2Ø63
        LDU #17574
        LDB #5
SCPRZ   LDA #3
SCPRC   PULU Y
        STY ,X++
        DECA
        BNE SCPRC
        LEAX 26,X
        DECB
        BNE SCPRZ
        LDA 18239
        LDB #5
        MUL
        ADDD #17724
        TFR D,U
        LDX #2Ø7Ø
        LDB #5
SCPRD   PULU A
        STA ,X
        LEAX 32,X
        DECB
        BNE SCPRD
        JSR 3ØØØØ
        RTS
        NOP
        NOP
PRSC    PSHS D,X,Y
        LDX #1824Ø
        LDB #6
        LDY #1814
PRSCB   LDA ,X
        PSHS X,B
        BITB #1
        BNE ROLL
        LDB #5
        MUL
        ADDD #17724
        TFR D,X
        PSHS Y
        LDB #5
PRSCA   LDA ,X+
        STA ,Y
        LEAY 32,Y
        DECB
        BNE PRSCA
        PULS Y
        LEAY 1,Y
ROLRET  PULS B,X
        LEAX 1,X
        DECB
        BNE PRSCB
        PULS Y,X,D
        RTS
ROLL    LDB #5
        MUL
        ADDD #17724
        TFR D,X
        PSHS Y
        LDB #5
RLLA    LDA ,X+
        PSHS A
        ANDA #15
        LSLA
        LSLA
        LSLA
        LSLA
        ORA #5
        STA 1,Y
        PULS A
        LSRA
        LSRA
        LSRA
        LSRA
        ORA #$5Ø
        STA ,Y
        LEAY 32,Y
        DECB
        BNE RLLA
        PULS Y
        LEAY 2,Y
        BRA ROLRET
```

THE SCREENING

The first thing this routine does is jump to the subroutine at $4AA5 which scrolls on the screen and prints up the sun.

Then X is loaded with the screen position which you want to start printing with the word 'SCORE'. U, the user stack pointer, is loaded with the start position of the word data in memory. This was input as a data table in an earlier part. And B is set to 5. The letters are only five bytes deep.

There are five letters in the word 'SCORE' plus a space, making six. But A is only loaded with 3 as the Y register is going to be used to print two bytes at a time.

The last two bytes of data are pulled off the stack—which is, in fact, the area of the data table pointed to by U. These two bytes are stored at the screen position pointed to by the X register and the one next to it. The X register is then incremented twice to move it onto the next screen position.

A is then decremented and the BNE SCPRI branches back until all six bytes of data, which make up a single line of the word, have been printed. When one line has been printed, LEAX 26,X increments the X register by 26 to move it from the right-hand end of one line to the left-hand end of the one below it. There are 32 locations across the screen and six letters: $32-6=26$.

B is then decremented and the processor branches back to deal with the next line if all 5 lines of the word have not been printed yet.

When they have, JSR PRSC jumps off to the subroutine that prints the score on the screen.

THAT'S LIVES

The next routine which prints up the word 'LIVES' is almost an exact repeat of the routine that printed 'SCORE' above—only the screen pointer in X and the data pointer in U are loaded with different values. The word is to be printed in a different position, obviously, and its data is in a different part of the data table. How the printing is done, though, is exactly the same.

This routine does not jump to the subroutine to print up the actual number of lives though. That follows on directly after the end of the word-print routine.

The number of lives Willie has left is stored in 18,239. The contents of this location are loaded into A. B is loaded with 5 and the contents of the two registers are multiplied together.

The display data for the figure has to be located in the data table before it can be printed on the screen. Each figure requires five bytes of data to describe it, so to locate the

beginning of the right area of data you have to count along the data table in multiples of five.

The result of a MUL—which multiplies the contents of A and B—is put in D. 17,724—the base address of the table—is then added to this, so the result points to the start of the appropriate figure's display data in memory.

This pointer is then transferred into the user stack pointer, U, so that this area of data effectively becomes the user stack.

Again X is loaded with the screen position where the number of lives is to be printed and B is loaded with 5, to count the five bytes of data needed to make up the figure. The first byte to figure display data is pulled off the user stack into A. Then it is stored at the screen position given by X.

X is incremented by 32 this time, as only one figure is to be printed, and the screen pointer has to be moved down one screen position to fill in the next line of pixels of the figure.

B is decremented and the processor branches back to deal with the next byte if all five bytes haven't been printed on the screen yet.

Then, to finish off, the processor jumps to the subroutine at 3Ø,ØØØ which plays the tune. And when it returns here, the RTS sends it back to BASIC.

This RTS will be overwritten later, when the whole program is put together. It is followed by two NOPs—which are No OPeration instructions. These do nothing and are only here to make enough room for a JSR instruction, when the RTS is overwritten. Then the processor will have to be directed to the action loop driving the whole program.

THE NUMBERS GAME

The data for each pixel line of the figures takes up a whole byte. So if you print them directly next to each other, they tend to merge into each other and become illegible.

To overcome that difficulty, the score print routine rolls figures along half a byte to create a gap between them and make the score readable.

The PRSC routine begins with the contents of D,X,Y being pushed onto the hardware stack. This is to preserve their values. In fact, it is only the value in Y that is going to be needed later. But when you are writing machine code it is best to push the contents of all the registers you are going to use in a subroutine. You might decide that you need to carry an important parameter in a register later. So the rule is—if in doubt, push it.

Memory location 18,24Ø holds the first byte of the score. The decimal value of each of the six digits of the score—from the highest to the lowest—are held in six memory locations from 18,24Ø up memory to 18,246.

There are six figures to be printed so B is loaded with 6. Y is loaded with the print position.

A is loaded with the contents of the location pointed to X, which points to the score memory. Then the values in X and B are preserved by pushing them onto the stack.

Bit zero of B is then checked. If B is even, that is, bit zero is not set, BNE ROLL branches off to the ROLL routine, which shoves every other figure half a space to the right. But if B is odd and bit zero is set, the processor continues with the next instruction.

To print the figure up on the screen, you have to proceed in the same way as printing the lives. You multiply the figure required by 5 and add the base address.

This time, though, the resulting pointer is transferred into X and the print position pointer in Y is stored by pushing it onto the stack.

The relevant byte of display data for the figure in question is then loaded into A and X is incremented. And it is then stored at the screen position pointed to by Y. Y is incremented by 32 to move down one line of pixels. B is decremented and the processor branches back to deal with the next byte of data until all five have been output to the screen.

When it has finished the initial screen position pointer is pulled off the stack again. It is incremented to move it onto the next print position to the right.

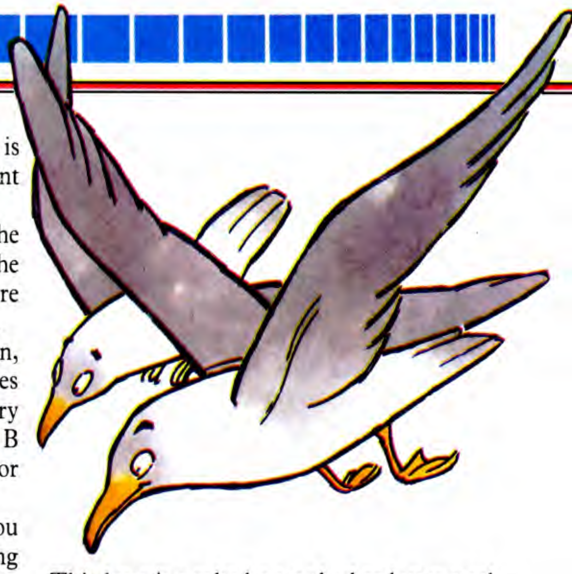
The values of counter in B and the score memory pointer in X are then pulled off the stack again. X is incremented ready to deal with the next figure to the right. And B is decremented, to count down the figures that have been dealt with. The processor then branches back to deal with the next figure, if they haven't all been dealt with yet.

If all the figures have been printed on the screen, the Y,X and D registers are restored by pulling the values they had at the beginning of the subroutine off the stack again. Then the processor returns to the place it was called in the main routine.

ROLLING ALONG

If the figure in B was even and the ROLL routine was called, the processor starts off by locating the start of the appropriate figure's data in the data table. It multiplies the score figure by five and adds on the base address, transfers the result into X, pushes the print position and loads the B register with the byte counter.

LDA,X+ loads the accumulator with a byte of the figure data and increments the pointer.



This byte is pushed onto the hardware stack to preserve it.

The contents of A are then ANDed with 15, which preserves the low nybble and clears the high one. Four LSLAs—Logic Shift Lefts on the Accumulator—move the low nybble a bit at a time into the high nybble.

ORing the result with 5 puts yellow—the background colour—into the low nybble. Then it is stored at the print position given by the screen pointer Y plus 1. This prints the right-hand half of the bit pattern on the screen at the position two to the right of the preceding figure—but shoved half a space to the left.

The whole bit pattern is then pulled off the hardware stack again and the high nybble is shifted right, into the low nybble. ORing with 5Ø hex sets the high nybble to bright yellow. And the resulting byte is stored at the print position pointed to by Y. This prints the left-hand side of the bit pattern in the right-hand half of the print position following the preceding figure—and marries it up to the right-hand half of the bit pattern in the left-hand half of the next print position. The other two halves of the print positions are filled in with the background colour yellow, effectively leaving a half digit gap between each figure.

LEAY 32,Y increments Y by 32 to deal with the next line of pixels down the figure. B is decremented and the processor branches back to pick up the next byte of the bit pattern, unless all five lines of pixels have already been printed.

When all that has been done, the processor drops out of the loop and increments Y by a further 2. This moves the screen pointer two positions to the right—effectively the even-numbered figure has occupied two adjacent screen positions.

BRA ROLRET then branches always back to the label ROLRET, where the pointers are updated before the routine jumps back to deal with the next—odd-numbered—digit.

FOX AND GEESE GAME-3

Allow your computer to collect its thoughts before the game commences. With these routines it can play fox or geese, and can look ahead to improve its play

This is the third and final part of the Fox and Geese article. The remaining routines are those which will enable the computer to take the part of either the fox or the geese. In fact the machine can be set to play both parts, when it competes against itself, or you could set it to play neither and play against one of your friends using the computer instead of a board and pieces.

```
210 LET P=B(G(1))+B(G(2))+B(G(3))+
    B(G(4))
220 LET X=F: IF P<B(32) THEN LET
    P=P-BX: LET X=33-F
230 LET P=P*B(X): RETURN
250 LET F=FN A(ABS P)-30: LET
    B=P/B(F): IF B<0 THEN LET
    B=B+BX: LET F=33-F
260 FOR A=1 TO 4: LET G(A)=FN
    A(B)+1: LET B=B-B(G(A)): NEXT A:
    RETURN
```

```
210 P=B(G(1))+B(G(2))+B(G(3))+
    B(G(4))
220 X=F:IFP<B(31)THENP=P-BX:
    X=31-F
230 P=P*B(X):RETURN
250 F=FNA(ABS(P))-31:B=P/B(F):
    IFB<0THENB=B+BX:F=31-F
260 FORA=1TO4:G(A)=FNA(B):
    B=B-B(G(A)):NEXTA:RETURN
```

```
210 P=B(G(1))+B(G(2))+
    B(G(3))+B(G(4))
220 X=F:IF P<B(31) THEN
    P=P-BX:X=31-F
230 P=P*B(X):RETURN
250 F=FNA(ABS(P))-31:B=P/B(F):IF
    B<0 THEN B=B+BX:F=31-F
260 FORA=1TO4:G(A)=FNA(B):B=B-B
    (G(A)):NEXTA:RETURN
```

```
210 P=B(G(1))+B(G(2))+B(G(3))+
    B(G(4))
220 X=F:IFP<B(31) THENP=P-BX:
    X=31-F
230 P=P*B(X):RETURN
250 F=FNA(ABS(P))-31:B=P/B(F):
    IFB<0 THENB=B+BX:F=31-F
260 FORA=1TO4:G(A)=FNA(B):
    B=B-B(G(A)):NEXT:RETURN
```

These are two of the most important routines in the program. The routine from Line 210 to Line 230 evaluates the playing position and packs it into a single number. Conversely, when the program has decided on the value of the best move, it has to convert that number into a move—or set of positions for the pieces. Lines 250 and 260 unpack the single value into the set of values needed for positioning the pieces.

These subroutines are called very frequently during the program, so they have been placed right at the start of the program.



FOX'S MOVE

```
1010 GOSUB 210
1020 GOSUB 310: GOSUB 250
1030 IF F>28 THEN PRINT AT 21,0;
    "THE FOX HAS WON□□
    □□□□□□□□□□□□□":
    GOTO 1410
1032 GOSUB 410: IF V=H THEN PRINT
    "THE GEESE HAVE WON□□
    □□□□□□□□□□□":
    GOTO 1410
1040 IF PF THEN GOTO 1110
1050 INPUT "MOVE FOX TO ";B: IF B=-1
    THEN GOSUB 2710: GOTO 1030
```



```
1060 FOR A=1 TO X(F): LET X=M(A,F): IF
    X=B THEN IF NOT (FN X(X)) THEN LET
    F=B: GOSUB 210:
    GOTO 1200
1070 NEXT A: PRINT AT 21,0;"THAT
    IS NOT LEGAL□□□□□
    □□□□□□□□□□": GOTO 1050
1110 LET L=SF: LET M=SF: LET
    V(M)=E*M: IF M>4 THEN DIM
    R(HF+3): DIM S(HF+3)
1112 GOSUB 1120: GOTO 1200
1120 IF L=1 THEN GOTO 410
1122 IF L<M-2 THEN GOSUB 1610: IF
    V<>0 THEN RETURN
1130 LET L=L-1: LET V(L)=H*L: LET
    A(L)=X(F): LET F(L)=F
1140 LET F=M(A(L),F(L))
1150 IF FN X(F)=0 THEN GOSUB 1320: IF
```

- PLAYING AT HIGHER LEVELS
- FOX'S MOVE ROUTINE
- MOVING THE GEESE
- CHOOSING THE BEST MOVE IN A 'ONE MOVER'

- PLAYING AT HIGHER LEVELS
- USING THE ALPHA-BETA ALGORITHM
- USING THE HASH-CODE TABLES FOR A FASTER GAME

```
   V>V(L) THEN LET V(L)=V: LET P(L)=F:
   IF V>V(L+1) THEN LET F=F(L): LET
   L=L+1: RETURN
1160 LET A(L)=A(L)-1: IF A(L)>Ø THEN
   GOTO 114Ø
1170 LET V=V(L): LET F=F(L): LET
   L=L+1: IF L=M THEN LET
   F=P(M-1): GOSUB 21Ø: RETURN
1172 IF L<M-2 THEN GOSUB 151Ø:
   RETURN
1180 RETURN
```



```
1Ø1Ø GOSUB 21Ø
1Ø2Ø GOSUB 31Ø:GOSUB 25Ø
1Ø3Ø IFF>27THEN PRINTTAB(8);
   "THE FOX HAS WON":GOTO141Ø
```



```
1Ø32 GOSUB41Ø:IFV=HTHEN PRINT
   TAB(8);"THE GEESE HAVE
   WON":GOTO141Ø
1Ø4Ø IF PF THEN PRINTTAB(8);
   "THINKING...":GOTO 111Ø
1Ø5Ø INPUT"▮▮▮▮▮▮▮▮▮
   MOVE FOX TO";B
1Ø55 GOSUB 34Ø:IF B=-1 THEN
   PRINTTAB(8);"THE GEESE HAVE
   WON":GOTO141Ø
1Ø6Ø FORA=ØTOX(F):X=M(A,F):IFX=B
   THENIFNOTFNX(X)THENF=B:GOSUB21Ø:
   GOTO12ØØ
1Ø7Ø NEXTA:PRINTTAB(8);"THAT IS NOT
   LEGAL":GOTO1Ø5Ø
111Ø L=SF:M=SF:V(M)=E*M:IFM>4
   THENFORA=ØTOHF:R(A)=Ø:
   NEXTA
1112 GOSUB112Ø:GOTO12ØØ
112Ø IFL=1THEN41Ø
1122 IFL<M-2THENGOSUB161Ø:
   IFV<>ØTHEN RETURN
113Ø L=L-1:V(L)=H*L:A(L)=X(F):
   F(L)=F
114Ø F=M(A(L),F(L))
115Ø IFFNX(F)<>ØTHEN 116Ø
1155 GOSUB132Ø:IFV>V(L)THENV(L)=V:
   P(L)=F:IFV>V(L+1)THENF=F(L):
   L=L+1:RETURN
116Ø A(L)=A(L)-1:IFA(L)>=ØTHEN114Ø
117Ø V=V(L):F=F(L):L=L+1:IFL=M
   THENF=P(M-1):GOSUB21Ø:RETURN
1172 IFL<M-2THENGOSUB151Ø:RETURN
118Ø RETURN
```

```
1Ø1Ø GOSUB 21Ø
1Ø2Ø GOSUB 31Ø:GOSUB 25Ø
1Ø3Ø IF F>27 THEN PRINT TAB(8);"THE FOX
   HAS WON":GOTO141Ø
1Ø32 GOSUB41Ø:IF V=H THEN PRINT
   TAB(8);"THE GEESE HAVE
   WON":GOTO141Ø
1Ø4Ø IF PF THEN PRINT TAB(8);
   "THINKING...":GOTO 111Ø
1Ø5Ø INPUT TAB(8)"MOVE FOX TO□" B
1Ø55 GOSUB 34Ø
1Ø6Ø A=Ø:REPEATX=M(A,F):IFX<>B OR
   FNX(X) THEN A=A+1:UNTIL
   A>X(F):ELSE UNTIL TRUE:
   F=B:GOSUB21Ø:GOTO12ØØ
1Ø7Ø PRINT TAB(8);"THAT'S NOT LEGAL":
   GOTO 1Ø5Ø
111Ø L=SF:M=SF:V(M)=E*M:IFM>4
   THEN FOR A=ØTOHF:R(A)=Ø:NEXTA
1112 GOSUB 112Ø:GOTO 12ØØ
112Ø IFL=1 THEN 41Ø
1122 IF L<(M-2) THEN GOSUB 161Ø:
   IFV<>Ø THEN RETURN
113Ø L=L-1:V(L)=H*L:A(L)=X(F):
   F(L)=F
114Ø F=M(A(L),F(L))
115Ø IF FNX(F)=Ø THEN GOSUB 132Ø:IF
   V>V(L) THEN V(L)=V:P(L)=F:
   IFV>V(L+1) THEN F=F(L):L=L+1:
   RETURN
116Ø A(L)=A(L)-1:IFA(L)>=Ø THEN 114Ø
117Ø V=V(L):F=F(L):L=L+1:
   IFL=M THEN F=P(M-1):
   GOSUB21Ø:RETURN
1172 IF L<(M-2) THEN GOSUB151Ø:
   RETURN
118Ø RETURN
```

```
1Ø1Ø SCREEN1,Ø:GOSUB21Ø:GOTO1Ø3Ø
1Ø2Ø XX=FNXX(G):YY=FNYY(G):PUT
   (XX,YY)-(XX+19,YY+19),SQ,PSET:XX
   =FNXX(G(C)):YY=FNYY(G(C))+5:
   PUT(XX,YY)-(XX+19,YY+9),GS,PSET
1Ø3Ø CLS:IF F>27 THEN PLAYV$:
   PRINT@7,"THE FOX HAS
   WON":GOTO141Ø
1Ø32 GOSUB41Ø:IF V=H THENPLAYV$:
   PRINT@6,"THE GEESE HAVE
   WON":GOTO141Ø
1Ø4Ø LINE(18Ø,Ø)-(255,191),PRESET,
   BF:IF PF THENDRAW"BM18Ø,5ØC4"+
   TH$:GOTO111Ø
```



```
1Ø5Ø DRAW"BM18Ø,8ØC4"+MW$:XX=FNXX(F):YY=FNYY(F):GOSUB181Ø:B=4*INT(YY/2Ø):B=FNCN(B)
1Ø55 IF B=-1 THENPLAYV$:PRINT@6,"THE GEESE HAVE WON":GOTO141Ø
1Ø6Ø FORA=ØTOX(F):X=M(A,F):IFX=B AND NOTFNX(X) THENA=X(F):XX=FNXX(F):YY=FNYY(F):PUT(XX,YY)-(XX+19,YY+19),SQ,PSET:F=B:GOSUB21Ø:NEXT:GOTO12ØØ
1Ø7Ø NEXT:GOSUB5ØØØ:GOTO1Ø4Ø
111Ø L=SF:M=SF:V(M)=E*M:IFM>4 THENFORA=ØTOHF:R(A)=Ø:NEXT
1112 LF=F:GOSUB112Ø:XX=FNXX(LF):YY=FNYY(LF):PUT(XX,YY)-(XX+19,YY+19),SQ,PSET:GOTO12ØØ
112Ø IFL=1 THEN41Ø
1122 IFL<M-2 GOSUB161Ø:IF V<>Ø THEN RETURN
113Ø L=L-1:V(L)=H*L:A(L)=X(F):F(L)=F
114Ø F=M(A(L),F(L))
115Ø IFFNX(F)=Ø GOSUB132Ø:IFV>V(L) THENV(L)=V:P(L)=F:IF V>V(L+1) THEN F=F(L):L=L+1:RETURN
116Ø A(L)=A(L)-1:IF A(L)>= ØTHEN 114Ø
117Ø V=V(L):F=F(L):L=L+1:IFL=M THENF=P(M-1):GOSUB21Ø:RETURN
1172 IF L<M-2 GOSUB 151Ø
118Ø RETURN
```

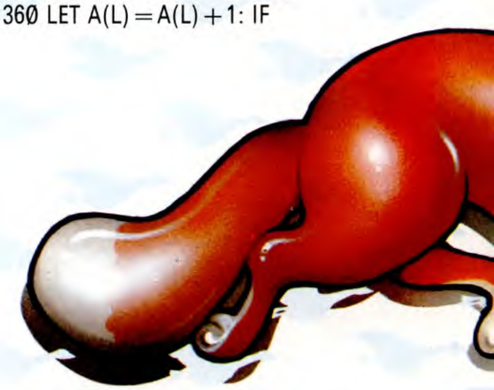
Lines 1Ø2Ø to 118Ø handle the fox's move. The routine uses the board display subroutine to display the current status of the board, then goes on to check if the fox has won in Line 1Ø3Ø. It then checks if there is at least one legal move open to the fox (or else the geese win) in Line 1Ø32.

If the player is controlling the geese, Lines 1Ø5Ø to 1Ø7Ø take the input and check its legality. If, on the other hand, the computer is controlling the fox, Lines 111Ø to 1112 look after the move. The number of plies that the program is looking at, M, and the current ply being considered, L, are set up in Line 111Ø. Lines 112Ø to 118Ø are a subroutine to evaluate the best move.

Four arrays are used in the best move subroutine: A contains the number of moves still to try; V, the best result so far; P, the move that yields that result; and F, the fox's previous position.

MOVING THE GEESE

```
5ØØ LET V=Ø: FOR C=1 TO 4: LET G=G(C): FOR A=1 TO Z(G): LET X=M(A,G): IF X<>F THEN IF NOT FN X(X) THEN RETURN
5Ø2 NEXT A: NEXT C: LET V=1: RETURN
12ØØ GOSUB 31Ø: GOSUB 25Ø
12Ø2 IF F>28 THEN PRINT AT 21,Ø;"THE FOX HAS WON□□□□□□□□□□□□□□□□□□": GOTO 141Ø
12Ø4 GOSUB 5ØØ: IF V THEN PRINT AT 21,Ø;"THE FOX HAS WON□□□□□□□□□□□□□□□□□": GOTO 141Ø
121Ø IF PG THEN GOTO 131Ø
122Ø INPUT "WHICH GOOSE TO MOVE ? ";G: IF G=-1 THEN GOSUB 271Ø: GOTO 12Ø2
123Ø LET C=FN Z(G): IF C=Ø THEN PRINT AT 21,Ø;"NO GOOSE AT ";G;"": GOTO 122Ø
124Ø INPUT "WHERE TO ";I: IF I=-1 THEN GOSUB 271Ø: GOTO 12Ø2
125Ø IF FN X(I) THEN PRINT AT 21,Ø;"NOT ONTO ANOTHER GOOSE□□□□□□□□□": GOTO 122Ø
126Ø IF I=F THEN PRINT AT 21,Ø;"NOT ONTO FOX□□□□□□□□□□□□□□□□□□□□□□": GOTO 122Ø
127Ø FOR A=1 TO Z(G): IF M(A,G)=I THEN LET G(C)=I: GOTO 1Ø1Ø
128Ø NEXT A: PRINT AT 21,Ø;"THAT IS NOT LEGAL□□□□□□□□□□□□□□□□□": GOTO 122Ø
131Ø LET L=SG: LET M=SG: LET V(M)=H*M: IF M>4 THEN DIM R(HF+3): DIM S(HF+3)
1312 GOSUB 132Ø: GOTO 1Ø2Ø
132Ø IF L=1 THEN GOTO 51Ø
1322 IF L<M-2 THEN GOSUB 161Ø: IF V<>Ø THEN RETURN
1324 LET L=L-1: LET V(L)=E*L: LET C=1
133Ø LET C(L)=C: LET F(L)=G(C): LET A(L)=1: IF A(L)>Z(G(C)) THEN GOTO 1362
134Ø LET B=M(A(L),F(L)): LET X=FN X(B): LET G(C)=B: IF X OR B=F THEN GOTO 136Ø
135Ø GOSUB 112Ø: LET C=C(L): IF V<V(L) THEN LET V(L)=V: LET P(L)=G(C)+C*32
1355 IF V<V(L) THEN LET G(C)=F(L): LET L=L+1: RETURN
136Ø LET A(L)=A(L)+1: IF A(L)<=Z(F(L)) THEN GOTO 134Ø
1362 LET G(C)=F(L): LET C=C+1: IF C<5 THEN GOTO 133Ø
137Ø LET V=V(L): LET L=L+1: IF L=M THEN LET C=INT (P(L-1)/32): LET G(C)=P(L-1)-C*32: GOSUB 21Ø: RETURN
1372 IF L<M-2 THEN GOSUB 151Ø: RETURN
138Ø RETURN
```



```
5ØØ V=-1:FORC=1TO4:G=G(C):IF Z(G)<ØTHENNEXT:RETURN
5Ø2 FORA=ØTOZ(G):X=M(A,G):V=VAND(FNX(X)ORX=F):NEXT:NEXT:RETURN
12ØØ GOSUB31Ø:GOSUB25Ø
12Ø2 IFF>27THEN PRINTTAB(8);"THE FOX HAS WON":GOTO141Ø
12Ø4 GOSUB5ØØ:IFVTHEN PRINT TAB(8);"THE FOX HAS WON":GOTO141Ø
```

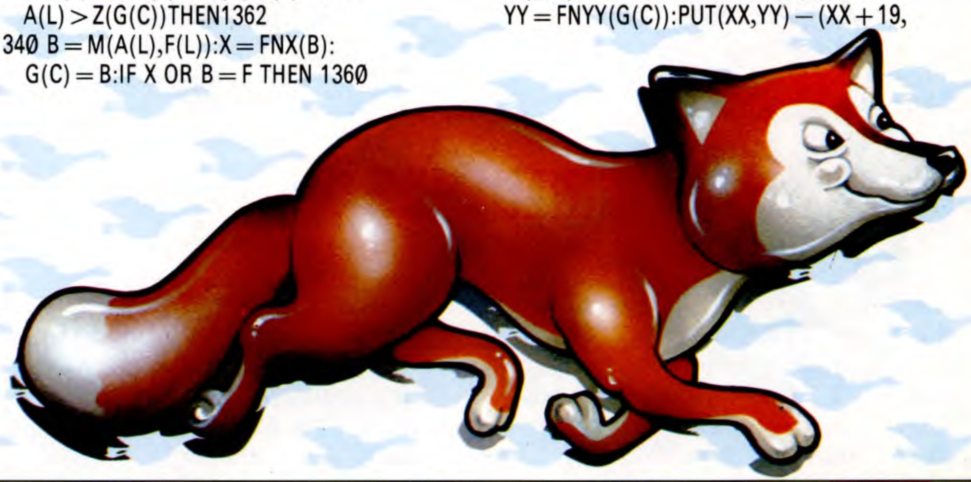
```
1210 IF PG THEN PRINTTAB(8);
   "THINKING...":GOTO 1310
1220 INPUT"■■■■■■■■
   WHICH GOOSE TO MOVE";G
1225 GOSUB 340
1230 C=FNZ(G):IFC=0THENPRINT
   TAB(8);"NO GOOSE AT";G:
   GOTO1220
1240 INPUT"■■■■■■■■
   WHERE TO";I
1245 GOSUB 340:IF I=-1 THEN
   PRINTTAB(8);"THE FOX HAS
   WON":GOTO1410
1250 IFFNX(I)THEN PRINTTAB(8);
   "NOT ONTO ANOTHER GOOSE":
   GOTO1220
1260 IFI=FTHEN PRINTTAB(8);
   "NOT ONTO FOX":
   GOTO1220
1270 FORA=-1TOZ(G):IFA>=0THENIF
   M(A,G)=ITHENG(C)=I:
   GOTO1010
```



```
1280 NEXTA:PRINT TAB(8);"THAT IS NOT
   LEGAL":GOTO1220
1310 L=SG:M=SG:V(M)=H*M:IFM>4
   THENFORA=0TOHF:
   R(A)=0:NEXTA
1312 GOSUB1320:GOTO1020
1320 IFL=1THEN510
1322 IFL<M-2THENGOSUB1610:IFV<>0
   THENRETURN
1324 L=L-1:V(L)=E*L:C=1
1330 C(L)=C:F(L)=G(C):A(L)=0:IF
   A(L)>Z(G(C))THEN1362
1340 B=M(A(L),F(L)):X=FNX(B):G(C)=B:
   IFXORB=FTHEN1360
1350 GOSUB1120:C=C(L)
1355 IFV<V(L)THENV(L)=V:P(L)=G(C)+
   C*32:IFV<V(L+1)THENG(C)=F(L):
   L=L+1:RETURN
1360 A(L)=A(L)+1:IFA(L)<=Z(F(L))THEN
   1340
1362 G(C)=F(L):C=C+1:IFC<5THEN1330
1370 V=V(L):L=L+1:IFL=MTHENC=INT
   (P(L-1)/32):G(C)=P(L-1)AND31:
   GOSUB210:RETURN
1372 IFL<M-2THENGOSUB1510:
   RETURN
1380 RETURN
```

```
500 V=-1:FOR C=1TO4:G=G(C):IF
   Z(G)<0 THEN NEXTC:RETURN
502 FOR A=0 TO Z(G):X=M(A,G):
   V=V AND (FNX(X) OR X=F):NEXTA:
   NEXTC:RETURN
1200 GOSUB310:GOSUB250
1202 IF F>27 THEN PRINT TAB(8);
   "THE FOX HAS WON":GOTO1410
1204 GOSUB 500:IF V THEN PRINT
   TAB(8);"THE FOX HAS WON":GOTO1410
1210 IF PG THEN PRINT TAB(8);
   "THINKING...":GOTO1310
1220 INPUTTAB(8)"WHICH GOOSE TO
   MOVE□" G:IF G=-1 THEN
   GOSUB2710: GOTO1202
1230 C=FNZ(G):IFC=0 THEN PRINT
   TAB(8);"NO GOOSE AT□";G:GOTO 1220
1240 INPUT TAB(8)"WHERE TO", I
1250 IFFNX(I) THEN PRINTTAB(8);
   "NOT ONTO ANOTHER GOOSE":
   GOTO 1220
1260 IFI=F THEN PRINT TAB(8);
   "NOT ONTO FOX":GOTO1220
1270 IF Z(G)>=0 THEN A=0:
   REPEAT:IF M(A,G)<>I THEN
   A=A+1:UNTIL A>Z(G):ELSE UNTIL
   TRUE:G(C)=I:GOTO1010
1280 PRINTTAB(8);"THAT'S NOT
   LEGAL":GOTO1220
1310 L=SG:M=SG:V(M)=H*M:
   IFM>4 THEN FOR A=0 TO HF:
   R(A)=0:NEXTA
1312 GOSUB1320:GOTO1020
1320 IFL=1 THEN510
1322 IF L<M-2 THEN GOSUB1610:
   IFV<>0 THEN RETURN
1324 L=L-1:V(L)=E*L:C=1
1330 C(L)=C:F(L)=G(C):A(L)=0:IF
   A(L)>Z(G(C))THEN1362
1340 B=M(A(L),F(L)):X=FNX(B):
   G(C)=B:IF X OR B=F THEN 1360
1350 GOSUB1120:C=C(L):IFV<V(L)THEN
   V(L)=V:P(L)=G(C)+C*32:IF
   V<V(L+1) THEN G(C)=F(L):L=L+1:
   RETURN
1360 A(L)=A(L)+1:IFA(L)<=Z(F(L))
   THEN1340
1362 G(C)=F(L):C=C+1:IF C<5 THEN
   1330
1370 V=V(L):L=L+1:IFL=M THEN
   C=INT(P(L-1)/32):G(C)=P(L-1) AND
   31:GOSUB210:RETURN
1372 IFL<M-2 THEN GOSUB1510:
   RETURN
1380 RETURN
```

```
500 V=-1:FORC=1TO4:G=G(C):IF
   Z(G)<0THENNEXT:RETURN
502 FORA=0TOZ(G):X=M(A,G):V=V AND
   (FNX(X)ORX=F):NEXTA,C:RETURN
1200 GOSUB250:XX=FNXX(F):
   YY=FNYY(F)+5:PUT(XX,YY)-(XX+19,
   YY+8),FX,PSET
1202 IF F>27THENPLAYV$:PRINT
   @6,"THE FOX HAS WON":
   GOTO1410
1204 GOSUB 500:IFV THEN PLAYV$:
   PRINT"THE FOX HAS WON":
   GOTO1410
1210 LINE(180,0)-(255,191),PRESET,BF:IF
   PG THENDRAW"BM180,50C4"+TH$:
   GOTO1310
1220 XX=FNXX(G(1)):YY=FNYY(G(1)):
   DRAW"BM180,80C2"+WG$:GOSUB1810
1225 G=4*INT(YY/20):G=FNCN(G)
1230 C=FNZ(G):IF C=0 GOSUB5000:
   GOTO1210
1240 DRAW"BM180,110C3"+MW$:GOSUB
   1810:I=4*INT(YY/20):I=FNCN(I)
1245 IF I=1 THENPLAYV$:PRINT@7,
   "THE FOX HAS WON":GOTO1410
1250 IFFNX(I) GOSUB5000:GOTO1210
1260 IFI=F GOSUB5000:GOTO1210
1270 FORA=-1TOZ(G):IFA>=0 THENIF
   M(A,G)=I THENXX=FNXX(G(C)):
   YY=FNYY(G(C)):PUT(XX,YY)-(XX+19,
```



```
   YY+19),SQ,PSET:G(C)=I:XX=FNXX(I):
   YY=FNYY(I):PUT(XX,YY+5)-(XX+19,
   YY+14),GS,PSET:A=Z(G):NEXT:
   GOTO1Ø1Ø
128Ø NEXT:GOSUB5ØØØ:GOTO121Ø
131Ø L=SG:M=SG:V(M)=H*M:IFM>4
   THENFORA=ØTOHF:R(A)=Ø:NEXT
1312 GOSUB132Ø:GOTO1Ø2Ø
132Ø IFL=1 THEN51Ø
1322 IFL<M-2 GOSUB161Ø:IF V<>Ø
   THEN RETURN
1324 L=L-1:V(L)=E*L:C=1
133Ø C(L)=C:F(L)=G(C):A(L)=Ø:IF
   A(L)>Z(G(C)) THEN1362
134Ø B=M(A(L),F(L)):X=FNX(B):G=G(C):
   G(C)=B:IF X ORB=F GOTO136Ø
135Ø GOSUB112Ø:C=C(L):IFV<V(L)THEN
   V(L)=V:P(L)=G(C)+C*32:IF
   V<V(L+1)THENG=G(C):G(C)=F(L):
   L=L+1:RETURN
136Ø A(L)=A(L)+1:IFA(L)<=Z(F(L))
   THEN134Ø
1362 G=G(C):G(C)=F(L):C=C+1:IFC<5
   THEN133Ø
137Ø V=V(L):L=L+1:IFL=M THENC=INT
   (P(L-1)/32):G=G(C):G(C)=P(L-1)
   AND31:GOSUB21Ø:RETURN
1372 IFL<M-2 GOSUB151Ø
138Ø RETURN
```

The routine from Line 12ØØ to Line 138Ø handles the geese. There are subroutines for when the player controls the geese—lines 122Ø to 129Ø—and when the computer controls the geese—Lines 132Ø to 138Ø.





BEST MOVES

```
41Ø LET V=H: FOR A=X(F) TO 1 STEP -1:
   LET X=M(A,F): IF FN X(X) THEN NEXT A:
   LET L=1: RETURN
42Ø LET B=F: LET F=X: GOSUB 21Ø: LET
   V=P: LET F=B: LET L=1: RETURN
51Ø LET V=E: FOR C=1 TO 4: LET
   G=G(C): IF -B(G)>V THEN GOTO 53Ø
52Ø FOR A=1 TO Z(G): LET X=M(A,G): IF
   FN X(X) OR (X=F) THEN NEXT A: GOTO
   53Ø
528 LET V=B(X)-B(G): LET D=C: LET
   B=X
53Ø NEXT C: LET G=G(D): LET G(D)=B:
   GOSUB 21Ø: LET V=P: LET G(D)=G:
   RETURN
```

```
41Ø V=H:FORA=X(F)TOØSTEP-1:
   X=M(A,F):IFFNX(X)<ØTHENNEXT:L=1:
   RETURN
42Ø B=F:F=X:GOSUB21Ø:V=P:F=B:
   L=1:RETURN
51Ø V=E:FORC=1TO4:G=G(C):IF
   -B(G)>VTHEN53Ø
52Ø FORA=ØTOZ(G):X=M(A,G):IFFNX(X)
   ORX=FTHENNEXT:GOTO53Ø
528 V=B(X)-B(G):D=C:B=X
53Ø NEXTC:G=G(D):G(D)=B:GOSUB21Ø:
   V=P:G(D)=G:RETURN
```

```
41Ø V=H:A=X(F):REPEAT:X=M(A,F):IF
   FNX(X)<Ø THEN A=A-1:UNTIL A<Ø:
   L=1:RETURN
42Ø UNTILTRUE:B=F:F=X:GOSUB 21Ø:
   V=P:F=B:L=1:RETURN
51Ø V=E:FOR C=1TO4:G=G(C):IF
   -B(G)>V OR Z(G)<ØTHEN 53Ø
52Ø A=Ø:REPEAT:X=M(A,G):IF FNX(X) OR
   (X=F) THEN A=A+1:UNTIL A>Z(G):
   GOTO53Ø
528 UNTILTRUE:V=B(X)-B(G):D=C:B=X
53Ø NEXTC:G=G(D):G(D)=B:GOSUB 21Ø:
   V=P:G(D)=G:RETURN
```

```
41Ø V=H:FORA=X(F)TOØ STEP-1:
   X=M(A,F):IFFNX(X)<Ø THENNEXT:L=1:
   RETURN
42Ø B=F:F=X:GOSUB21Ø:V=P:F=B:
   L=1:A=Ø:NEXT:RETURN
51Ø V=E:FORC=1TO4:G=G(C):IF
   -B(G)>V THEN53Ø
52Ø FORA=ØTOZ(G):X=M(A,G):IFFNX(X)
   OR X=F THENNEXT:GOTO53Ø
528 V=B(X)-B(G):D=C:B=X
53Ø NEXT:G=G(D):G(D)=B:GOSUB21Ø:
   V=P:G(D)=G:IFSG=1 THENG(D)=B:
   C=D
54Ø RETURN
```

These subroutines are concerned with the 'one mover'—in other words, at this stage the computer is only looking one move ahead when searching for the best move.

Lines 41Ø and 42Ø go through all the possible moves open to the fox, using the map array M, set up in the subroutine starting at Line 211Ø (see earlier). The subroutine returns a value of P, configuration after best move, and V, evaluation after best move.

Lines 51Ø and 53Ø are a similar routine, but this time it looks for the best move for the geese. P and V are set in the same way as in the previous subroutine.

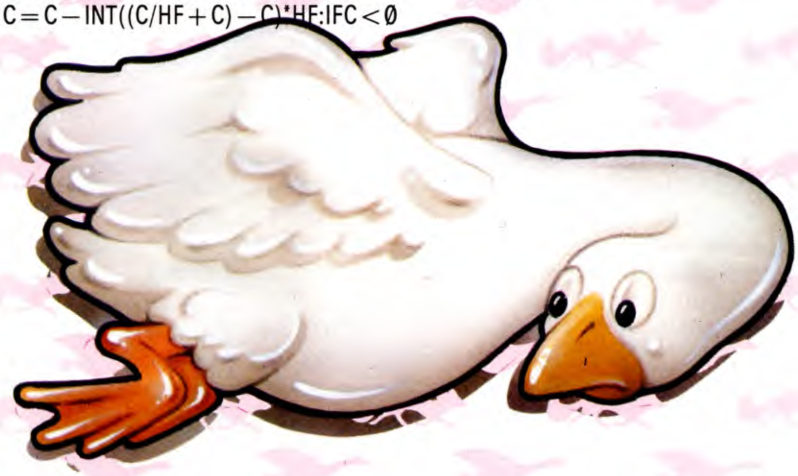
In either case, GOSUB 21Ø picks the best move from those evaluated.

THE HASH CODE TABLE

```
151Ø GOSUB 21Ø: LET C=P
152Ø LET C=C-INT ((C/HF+C)-C)*HF:
   IF C<Ø OR C>=HF THEN GOTO 152Ø
155Ø FOR A=C+1 TO C+4: IF R(A)<>Ø
   AND R(A)<>P THEN NEXT A: RETURN
156Ø LET R(A)=P: LET S(A)=V: RETURN
```

```
1610 GOSUB 210: LET C=P
1620 LET C=C-INT ((C/HF+C)-C)*HF:
   IF C<Ø OR C>=HF THEN GOTO 1620
1650 FOR A=C+1 TO C+4: IF R(A)<>Ø
   AND R(A)<>P THEN NEXT A: LET
   V=Ø: RETURN
1660 LET V=S(A)*(R(A)=P): RETURN
```

```
1510 GOSUB210:C=P
1520 C=C-INT((C/HF+C)-C)*HF:IFC<Ø
   ORC>=HFTHEN1520
1550 FORA=CTOC+4:IFR(A)<>ØAND
   R(A)<>PTHENNEXT:RETURN
1560 R(A)=P:S(A)=V:RETURN
1610 GOSUB210:C=P
1620 C=C-INT((C/HF+C)-C)*HF:IFC<Ø
   ORC>=HFTHEN1620
1650 FORA=CTOC+4:IFR(A)<>ØAND
   R(A)<>PTHENNEXT:V=Ø: RETURN
1660 V=-S(A)*(R(A)=P): RETURN
2500 DIM(1999), S(1999)
```



```
1510 GOSUB 210:C=P
1512 IF ABS(C/HF)>1E9 THEN
   C=C-((C/HF+C)-C)*HF:
   GOTO1512
1520 C=C-INT((C/HF+C)-C)*HF:
   IF C<Ø OR C>=HF THEN 1520
1550 A=C:REPEAT:IF R(A)<>Ø AND
   R(A)<>P THEN A=A+1:UNTIL
   A=C+4:RETURN:ELSE UNTIL TRUE
1560 R(A)=P:S(A)=V:RETURN
1610 GOSUB 210:C=P
1612 IF ABS(C/HF)>1E9 THEN
   C=C-((C/HF+C)-C)*HF:
   GOTO1612
1620 C=C-INT((C/HF+C)-C)*HF:
   IF C<Ø OR C>=HF THEN 1620
1650 A=C:REPEAT:IF R(A(<>Ø AND
   R(A)<>P THEN A=A+1:UNTIL
   A=C+4:V=Ø:RETURN:ELSE UNTIL
   TRUE
1660 V=-S(A)*(R(A)=P):RETURN
```

```
1510 GOSUB210:C=P
1520 C=C-INT((C/HF+C)-C)*HF:IFC<Ø
   ORC>=HF THEN1520
1550 FORA=C TOC+C:IFR(A)<>ØAND
   R(A)<> PTHENNEXT:RETURN
1560 R(A)=P:S(A)=V:A=C+C:NEXT:
   RETURN
1610 GOSUB210:C=P
1620 C=C-INT((C/HF+C)-C)*HF:IFC<Ø
   ORC>=HF THEN1620
1650 FORA=C TOC+C:IFR(A)<>Ø AND
   R(A)<>P THENNEXT:V=Ø:RETURN
1660 V=-S(A)*(R(A)=P):A=C+C:NEXT:
   RETURN
```

```
1810 SCREEN1,Ø:PUT(XX,YY)-(XX+19,
   YY+19),SQ,NOT:FORZ=1TO100:NEXT
1820 PUT(XX,YY)-(XX+19,YY+19),SQ,
   NOT
1830 K$=INKEY$:IF K$="↑" AND YY>8
   AND XX>8 THEN YY=YY-2Ø:XX=
   XX-2Ø:GOTO181Ø
1840 IF K$=CHR$(1Ø) AND YY<129 AND
   XX<129 THEN YY=YY+2Ø:
   XX=XX+2Ø:GOTO181Ø
1850 IF K$=CHR$(8) AND XX>28 THEN
   XX=XX-4Ø:GOTO181Ø
1860 IF K$=CHR$(9) AND XX<128
   THENXX=XX+4Ø:GOTO181Ø
1870 IF K$=CHR$(13) THEN RETURN
1875 IF K$="Q" THEN YY=Ø:XX=-12:
   RETURN
1880 GOTO 181Ø
```

In the higher levels of play, you'll want to apply the alpha-beta algorithm—see page 1098. In fact, you've already entered the algorithm as part of Fox's move and Geese move routines. Before the algorithm is applied, they check if it is appropriate to use the algorithm—is the chosen level of play sufficiently high to warrant its use?

The routine from Line 111Ø to Line 115Ø looks after the fox, while the routine from Line 131Ø to 135Ø looks after the geese.

The algorithm is applied in the last IF test at the end of Lines 115Ø and 135Ø, after V(M) has been set to the appropriate level in Lines 111Ø and 131Ø.

The alpha-beta algorithm is most efficient if the computer considers what are likely to be the best moves first—for the geese, the highest-numbered square in each row of four squares, and for the fox, the square open to it that is nearest the geese end.

The alpha-beta algorithm is used in conjunction with a technique known as *hash-coding* to build up and use a table of the moves already considered. Hash-coding allows the computer to check quickly if the move has already been considered. The larger the hash-code table that can be built into the program, the faster-running that program should be.

The table is initialized in Lines 25ØØ, 275Ø and 28ØØ. There are theoretical 'best values' for the dimensions of the arrays holding the tables—Line 25ØØ. The arrays have been DIMensioned to as large as possible, given the available RAM in each of the machines.

The table is zeroed in Lines 111Ø and 131Ø, the contents are checked in Lines 1122 and 1322, and set in Lines 1172 and 1372. The checking subroutine starts at Line 161Ø, and the setting subroutine starts at Line 151Ø. BBC owners with disk filing systems should enter the following before RUNning.



```
*TAPE
FOR A%=ØTO&1600:?(&EØØ+A%)=?
   (PAGE+A%):NEXT
PAGE=&EØØ
OLD
RUN
```

The Dragon/Tandy game uses a flashing cursor to move the pieces. Move it to the square or piece you wish and press RETURN in response to the prompts to the right of the screen.

PREDICTING THE UNPREDICTABLE

What are the chances of your selection scoring a big win on the football Pools this weekend? You might not improve your luck, but may understand why you've lost

Instructing a novice at the controls of a modern jetliner or a fighter-bomber would be wastefully expensive, when factors such as fuel, landing fees and back-up facilities are taken into account. This is why military authorities find it cheaper to expend large sums for trainer aircraft and simulators—computer controlled mock-ups that never leave the ground, but give a trainee a realistic impression of flying.

The same principle holds true in a large number of circumstances—including games programming, industrial development and marketing, scientific research and government. It is better to program a computer to predict the probable result of structural failure, or of a particular economic policy, for example, than to do the actual experiment, which might last five years or even more in some circumstances. Not surprisingly, simulations are a useful and favoured facility. But their complexity means that they were only really made possible by the advent of computers. Now, even the home micro can make light work of simulation.

One of the chief tools of such computer modelling is the set of rules provided by the mathematical study of probability. In this context, you may find it helpful to look back to the article on pages 694 to 700, which explains some of the theory behind this. The rest of the model comes from an analysis of statistical information, gathered from a survey of the real situation.

Given a few simple rules, the home micro user can conjure up alternative futures for a host of different events, but the reliability of the results depend on the accuracy of the data fed into the program or the precision of the rules by which the game is played. This fact can be demonstrated by a simulation of the outcome of football matches.

Every Saturday afternoon during the soccer season, millions of pools punters eagerly await the football results and the chance of a bumper first dividend. Sadly, all but a few are disappointed, and must wait for another week and another chance. If you don't want to wait for a whole week, then key in the first program and try your luck as many times as you like:

```
10 POKE 23658,0
20 DIM a(55): DIM r$(4,14)
30 BORDER 0: PAPER 0: INK 7: CLS
50 PRINT AT 0,7; INVERSE 1;
   "□□TREBLE CHANCE□□"
60 PRINT : PRINT
80 PRINT "□□HOW MANY SELECTIONS
   FROM 55□□□□□MATCHES DO YOU
   WISH TO MAKE□□□□□□□□□□
   □□□□□(8-16)"
90 INPUT n
95 IF n<1 OR n>16 THEN CLS : GOTO 80
100 PRINT : PRINT "□WHAT ARE THE
   NUMBER OF MATCHES□□THAT YOU
   WISH TO SELECT (1-55)"
120 FOR k=1 TO n
130 INPUT a(k)
140 NEXT k
150 CLS
160 PRINT FLASH 1; PAPER 2;AT
   10,6;"□MATCHES IN PROGRESS□"
170 FOR v=1 TO 2000: NEXT v
180 CLS
190 PRINT AT 0,10; INVERSE 1;
   "□□RESULTS□□"
200 LET r$(1)="HOME WIN": LET
   r$(2)="AWAY WIN"
210 LET r$(3)="GOAL-LESS DRAW": LET
   r$(4)="SCORE DRAW"
220 FOR i=1 TO n
230 LET y=RND*1
240 IF y<=.5 THEN LET z$=r$(1)
250 IF y>.5 AND y<=.75 THEN LET
   z$=r$(2)
260 IF y>.75 AND y<=.9 THEN LET
   z$=r$(3)
270 IF y>.9 THEN LET Z$=r$(4)
280 PRINT "MATCH NUMBER";TAB
   14;a(i);TAB 18;z$
290 NEXT i
300 PRINT : PRINT
310 INPUT "DO YOU WANT ANOTHER GO
   ?□";t$
320 IF t$="n" THEN GOTO 340
330 GOTO 30
340 STOP
```

```
20 DIM A(55),R$(4)
50 PRINT "♡■>■■TREBLE CHANCE"
80 PRINT"■■■HOW MANY SELECTIONS
   FORM 55 DO YOU":PRINT"WISH TO
   MAKE"
90 INPUT"(8-16)";N
100 PRINT "■■WHAT ARE THE NUMBERS
   OF THE MATCHES"
110 PRINT "THAT YOU WISH
   TO":PRINT"SELECT (1-55)"
120 FOR K=1 TO N
130 INPUT A(K)
140 NEXT K
160 PRINT "♡■>■■MATCHES IN
   PROGRESS"
170 FOR V=1 TO 2000:NEXT V
190 PRINT "♡■>■■RESULTS"
200 R$(1)="HOME WIN":
   R$(2)="AWAY WIN"
210 R$(3)="GOAL-LESS DRAW":
   R$(4)="SCORE DRAW"
220 PRINT "■MATCH":FOR I=1 TO N
230 Y=RND(1)
240 IF Y<=0.5 THEN Z$=R$(1)
250 IF Y>0.5 AND Y<=0.75 THEN
   Z$=R$(2)
260 IF Y>0.75 AND Y<=0.9 THEN
   Z$=R$(3)
270 IF Y>0.9 THEN Z$=R$(4)
280 PRINT A(I)TAB(6)Z$
290 NEXT I
310 PRINT"■DO YOU WANT ANOTHER GO
   (Y/N)?";
320 GET T$:IF T$="N" THEN
   PRINT"♡":END
330 IF T$="Y" THEN RUN
340 GOTO 320
```

```
20 DIM A(55),R$(4)
50 MODE1:PRINTTAB(14,3)
   "TREBLE CHANCE"
80 PRINT''"HOW MANY SELECTIONS FROM
   55 MATCHES DO□□YOU WISH TO
   MAKE (8-16)?"
90 INPUT N
100 PRINT"WHAT ARE THE NUMBERS OF
   THE MATCHES''"THAT YOU WISH TO
   SELECT (1-55)?"
120 FOR K=1 TO N
130 INPUTA(K)
140 NEXT
```

- RULES OF THE GAME
- SCORING
- DECIDING RESULTS
- SELECTING NUMBERS FROM A HAT OR BOX
- GENERATING RANDOM NUMBERS
- TYPES OF RANDOM NUMBER
- SAMPLES AND SURVEYS
- SAMPLING
- SINGLE-PASS SEARCH

```
16Ø PRINTTAB(11)"MATCHES
   IN PROGRESS"
17Ø D=INKEY(5ØØ)
19Ø CLS:PRINTTAB(17,3)
   "RESULTS"
2ØØ R$(1)="HOME WIN":
   R$(2)="AWAY WIN"
21Ø R$(3)="GOAL-LESS DRAW":
   R$(4)="SCORE DRAW"
22Ø FOR I=1 TO N
23Ø Y=RND(1)
24Ø IF Y<=.5 THEN
   Z$=R$(1)
25Ø IF Y>.5 AND Y<=.75 THEN
   Z$=R$(2)
26Ø IF Y>.75 AND Y<=.9 THEN
   Z$=R$(3)
27Ø IF Y>.9 THEN
   Z$=R$(4)
28Ø PRINT"MATCH NUMBER□";
   A(I)TAB(2Ø)Z$
29Ø NEXT
31Ø INPUT'"DO YOU WANT ANOTHER
   GO",G$
32Ø IF G$="N" THEN END
33Ø GOTO 5Ø
```




```
2Ø DIM A(55)
3Ø CLS
5Ø PRINT@9,"treble chance"
6Ø PRINT:PRINT
8Ø INPUT"HOW MANY SELECTIONS FROM
   55□□□□□MATCHES DO YOU WISH
   TO MAKE□□□□□(8-13)";N
9Ø IF N<8 OR N>13 THEN 8Ø
1ØØ PRINT"WHAT ARE THE NUMBERS OF
   THE□□□□□MATCHES THAT YOU
   WISH TO SELECT (1-55)□"
11Ø T=Ø
12Ø FORK=1 TO N
13Ø INPUT A(K)
14Ø NEXT K
15Ø CLS
16Ø PRINT@262,"matches in progress"
17Ø FOR V=1 TO 2ØØØ:NEXT V
18Ø CLS
19Ø PRINT@12,"results"
22Ø FOR I=1 TO N
23Ø Y=RND(Ø)
24Ø IF Y<=.5 THEN Z$="HOME WIN"
25Ø IF>.5 AND Y<=.75 THEN Z$="AWAY
   WIN"
26Ø IF Y>.75 AND Y<=.9 THEN
   Z$="GOAL-LESS DRAW"
```

```
27Ø IF Y>.9 THEN Z$=
    "SCORE DRAW"
28Ø PRINT"MATCH NUMBER ";
    A(I);TAB(17);Z$
29Ø NEXT I
3ØØ PRINT:PRINT
31Ø INPUT"DO YOU WANT ANOTHER GO
    (Y/N)□";T$
32Ø IF T$="N" THEN END
33Ø GOTO 3Ø
```

There are no first dividends in this football simulation, but it may sharpen your feel for the real thing. In any event, it will demonstrate just how small is the chance of getting 24 points.

RUN the program, and choose a number of matches between 8 and 16 in the range 1 to 55. This is equivalent to placing an X against the games chosen as likely draws. The results for the games you selected are then displayed on the screen. Scoring three points for a score draw, two for a no-score draw, one-and-a-half for an away win and one for a home win, the best eight matches in a typical selection would yield a total of only about 15 points—not a good start.

In January and February, bad weather sometimes forces matches to be cancelled. When this happens, a panel of football experts decides how the results would have turned out if the games had actually taken place. In fact, both the program and the panel are simulators—they provide a symbolic representation of a real process.

The program is not particularly difficult. After matches have been selected (Lines 12Ø to 14Ø) the main part of the simulation (Lines 22Ø to 29Ø) decides the outcome of each match, and prints the results.

DECIDING RESULTS

The basis on which the computer decides to print 'score draw', 'no-score draw', or another result, is the old pools punter's rule—that a half of the matches played turn out to be home wins, a quarter away wins, and the remaining quarter result in draws. It has been assumed that the number of ties is divided between score draws and no-score draws in the ratio two to three. This ratio gives factors of $\frac{2}{5}$ and $\frac{3}{5}$. When multiplied by $\frac{1}{4}$, the values are Ø.1 and Ø.15, so that the probability (see pages 694 to 700) of a score draw is 0.1.

It is worth noting that previously, each match had only three possible results (home win, away win and draw). Then, the total number of different possible outcomes of eight matches were three to the power eight, or 6561. The introduction of a score draw category increased this dramatically to (4↑8) or 65,536. If the chance of choosing a score draw is 1 in 10 (0.1), then the odds against picking eight score draws in eight matches is 1 to 100 million, which is most discouraging.

Choosing numbers in the treble chance is similar to deciding who wins a raffle. Usually, the tickets are placed in a large box or a hat, and someone, without looking, picks out the lucky number. The allocation of horses in the office Derby sweepstake can be organized in a similar fashion, and the method is termed top hat simulation.

Suppose a piece of paper is cut into four quarters. On the first quarter, the words 'away win' are written. 'Draw' is written on the second quarter and the other two pieces of paper bear the message 'home win'. Now, if the four pieces of paper are folded and placed in a top hat, picking a piece of paper at random is equivalent to deciding the outcome of a football match. The process would be just as effective if one each of the numbers 1, 2, 3 and 4 had been written on the four pieces of paper. The outcome could then have been decided by referring to a table:

Number drawn	Result
1, 2	home win
3	away win
4	draw

Similarly, when score draws are taken into account, if 40 pieces of paper bearing one each of the numbers 1, 2, ... 40 are placed in the top hat, the result of picking out a particular piece of paper can be intepreted according to the following table:

Number drawn	Result
1 to 20	home win
21 to 30	away win
31 to 36	no-score draw
37 to 40	score draw

The BASIC function RND, which generates random numbers, gives a computation equivalent to picking numbers out of a hat. You can specify that RND generates a number which is equally likely to take any value between 0 and 1. It only remains to rewrite the table in decimal format:

Value of RND	Result
from 0 to 0.5	home win
greater than 0.5 to 0.75	away win
greater than 0.75 to 0.90	no-score draw
greater than 0.90 to 1.00	score draw

These are the actual values that are coded at Lines 23Ø to 27Ø in the program above.



The treble chance simulation is only as good as the probability estimate used in the program. You could collect your own data from the Sunday papers, for example, and decide whether a greater proportion of score draws, say, is more usual.

GENERATING RANDOM NUMBERS

Many years ago, manual or mechanical methods, such as dice throwing, card-shuffling or the spinning of a roulette wheel, were used to generate random numbers. These methods were slow and tedious, so the famous American mathematician, John Von Neuman, proposed a mid-square technique. Starting with a four-digit number (the seed), the next 'random number' is obtained by multiplying the seed by itself, and then taking the middle four digits. For example, suppose the seed is 5272. Then the second number can



be generated by taking the middle four digits of (5272↑2) or 27,793,984. The answer (7939) is, practically, random. A second number can be obtained by squaring 7939, and so the process continues.

How, you may ask, can any mathematical process—which must be repeatable—produce true random numbers? The answer is that it can't. The numbers so obtained, however, behave as though they are random, and are usually referred to as pseudo-random or quasi-random. Numbers generated by using RND are, in fact, pseudo-random. Unfortunately, the mid-square technique is not very useful for generating random numbers by computer.

Besides being slow, the sequence quickly repeats, and once a zero is obtained, the whole process ends. Most home micros use a congruence method which uses remainders to generate pseudo-random sequences. The next program uses a simple formula and the INT (integer) function to provide an example of this:

```
20 BORDER Ø: INK 7: PAPER Ø: CLS
30 PRINT AT Ø,4; INVERSE 1;
   "□PSEUDO-RANDOM NUMBERS□"''
40 INPUT "□HOW MANY NUMBERS
   ?□";n: LET s=Ø
50 LET x=.677829*PEEK 23673/5Ø
60 LET x=x*1842.95
70 LET x=x-INT (x)
80 LET p=INT (x*1ØØØ): PRINT
   "□□□□□□";.ØØ1*p,
90 LET s=s+1
110 IF s<=n THEN GOTO 6Ø
120 STOP
```

```
30 PRINT"■■>■■PSEUDO
   -RND NUMBERS■"
40 PRINT"HOW MANY NUMBERS":
   INPUT N
50 X=.677829*TI/6Ø
60 X=X*1842.95
70 X=X-INT(X)
80 P=INT(X*1ØØØ):PRINT .ØØ1*P,
90 S=S+1
110 IF S<=N THEN 6Ø
```

```
30 MODE1:PRINTTAB(8,3)
   "PSEUDO-RANDOM NUMBERS"
40 INPUT"HOW MANY NUMBERS",N:
   S=Ø
50 X=.677829*(TIME/1ØØ-INT
   (TIME/1ØØ))
60 X=X*1842.95
70 X=X-INT(X)
80 P=INT(X*1ØØØ):PRINT.ØØ1*P;
90 S=S+1
110 IF S<=N THEN 6Ø
```

```
20 CLS
30 PRINT@3,"pseudo-random numbers"
40 INPUT"HOW MANY NUMBERS□";
   N:S=Ø
50 X=.677829*TIMER/5Ø
60 X=X*1842.95
70 X=X-INT(X)
80 P=INT(X*1ØØØ):PRINTLEFT$
   (STR$(P/1ØØØ)+"□□□□
   □□□",8);
90 S=S+1
110 IF S<=N THEN 6Ø
120 END
```

Line 5Ø uses the computer time function to set a different seed value for each run. The number .677829 is a fairly arbitrary constant. After the start value has been multiplied by another constant (Line 6Ø), the remainder or decimal part is obtained. Change the value of the constants at Lines 5Ø and 6Ø, and RUN the program again to see what sort of results you obtain.

For most purposes, it is wise—and far simpler—to use the RND function provided by your own computer. Varying the value of x in the expression RND(x) will usually enable you to select a repeatable sequence which is good for testing or to re-seed each time.

Perhaps the most important point to remember when writing programs is that the function RND provides a random variable and not an algebraic variable. RND(1)—or RND(Ø) on the Dragon and Tandy—listed at one line of a program will not take the same value as RND(1) listed elsewhere.

SAMPLES AND SURVEYS

Political opinion polls, consumer research organizations and governments use computers to generate random samples. It is important, for example, that a market research company interviewing 1000 people gets a typically varied sample, representative of a much larger group of the population. It would not do, for example, to choose people who were all members of a vintage car club, if the survey was supposed to check the national pattern of car ownership.

The best way to determine that a sample is representative is to pick them randomly—this excludes any particular bias.

This still holds true, even when you are working with a smaller sample who do have a particular common interest, like the members of a vintage car club, or the readers of *INPUT*. Selecting randomly is still important here, if, say, you wanted to determine the spread of computer ownership of particular computers amongst *INPUT* readers.

You may even wish to pick a sample from your own computerized club membership list. Basically, the sampling process is similar to a top-hat simulation in which several pieces of paper are picked from the hat. When simulating, however, each piece of paper is returned to the hat before the next is chosen. In sampling, once a piece of paper has been selected from the top hat, it is placed to one side before the next is chosen. Enter the next program to see how RND can be used to generate a random sample:

```
10 DIM b$(1Ø,16)
20 DIM a$(1Ø,16)
```

```
30 BORDER 0: PAPER 0: INK 7: CLS
50 PRINT AT 0,7; INVERSE 1;
   "□RANDOM SAMPLING□"
90 PAUSE 100: CLS
100 FOR i=1 TO 10: READ a$(i): NEXT i
110 INPUT "□SAMPLE SIZE ?□";n
120 FOR v=1 TO 10: LET b$(v)=a$(v):
   NEXT v
130 FOR j=1 TO n
140 LET r=1+INT (RND*10)
150 IF b$(r)="□□□□□□□□□
   □□□□□□□□" THEN GOTO 140
160 PRINT b$(r)
170 LET b$(r)=""
180 NEXT j
190 INPUT "□ANOTHER SAMPLE (y/n)
   ?□";g$
200 IF g$="y" THEN CLS : GOTO 110
210 STOP
220 DATA"BONN","COPENHAGEN",
   "LONDON"
230 DATA "MADRID","MOSCOW","NEW
   YORK"
240 DATA "PARIS","ROME","STOCKHOLM",
   "VIENNA"
```

```
20 DIM A$(10)
50 PRINT "♥■>■■RANDOM
   SAMPLING■■"
100 FOR I=1 TO 10:READ A$(I):
   NEXT I
110 INPUT "SAMPLE SIZE";N:PRINT
120 FOR V=1 TO 10:B$(V)=A$(V):
NEXT V
130 FOR J=1 TO N
140 R=1+INT(RND(1)*10)
150 IF B$(R)="" THEN 140
160 PRINT B$(R)
170 B$(R)=""
180 NEXT J
190 PRINT "■■ANOTHER SAMPLE
   (Y/N)?"
200 GET G$:IF G$="Y" THEN RUN
210 IF G$<>"N" THEN 200
215 PRINT"♥"
220 DATA BONN,COPENHAGEN,LONDON
230 DATA MADRID,MOSCOW,NEW YORK
240 DATA PARIS,ROME,STOCKHOLM,
   VIENNA
```

```
20 DIM A$(10),B$(10)
50 MODE1:PRINTTAB(13,3)"RANDOM
   SAMPLING"
90 D=INKEY(300)
100 FOR I=1 TO 10:READ A$(I):
   NEXT
110 INPUT""SAMPLE SIZE 1-10",N
120 FOR V=1 TO 10:B$(V)=A$(V):
   NEXT
130 FOR J=1 TO N
140 R=RND(10)
150 IF B$(R)="" THEN 140
160 PRINTB$(R)
170 B$(R)=""
180 NEXT
190 INPUT""ANOTHER SAMPLE (Y/N)",G$
200 IF G$="Y" THEN 110
210 END
220 DATA BONN,COPENHAGEN,LONDON
230 DATA MADRID,MOSCOW,
   NEW YORK
240 DATA PARIS,ROME,STOCKHOLM,
   VIENNA
```

```
40 CLS
50 PRINT@8,"random sampling":
   PRINT:PRINT
100 RESTORE:FOR I=1 TO 10:
   READ A$(I):NEXT I
110 INPUT"SAMPLE SIZE□";N:
   PRINT
115 IF N>10 THEN 40
120 FOR V=1 TO 10:B$(V)=A$(V):
   NEXT V
130 FOR J=1 TO N
140 R=1+INT(RND(0)*10)
150 IF B$(R)="" THEN 140
160 PRINTB$(R)
170 B$(R)=""
180 NEXT J
190 PRINT:INPUT"ANOTHER SAMPLE
   (Y/N)□";G$:PRINT:PRINT
200 IF G$="Y" THEN 110
210 END
220 DATA BONN,COPENHAGEN,
   LONDON
230 DATA MADRID,MOSCOW,
   NEW YORK
240 DATA PARIS,ROME,STOCKHOLM,
   VIENNA
```

Microtip

MAKING PREDICTIONS

The simulation techniques described in this article have many of the elements you would need to devise a method of predicting results of a game or to carry out a survey. So it should be easy for you to modify the programs, or use certain sections of them, in your own code. You could, for example, write a program to decide which matches will result in a score draw, then select eight of these matches at random and compare your predictions with actual results. This would require sections of the 'treble chance' and the 'random sampling' programs, together with a few lines to link them and keep track of your selections. One use for such a program would be to remove the tendency for a person to choose matches on the coupon according to the separation of the numbers—they spread their choice at almost regular intervals on the coupon. In fact it could be interesting to compare both methods of choosing which of the games will produce the appropriate results.

After the data have been read in (Line 100), a random integer R between 1 and 10 is generated (Line 140). Your computer will then print the Rth item (Line 160) on the list. Once an item has been selected, it must be removed from the data bank so that it is not selected a second time. Line 170 looks after that job. Notice that you could use RND(10) to generate a random variable in the range 1 to 10, but the method shown at Line 140 follows on from the use of decimal fractions to test the range of probabilities.

Usually, the data population from which the sample is drawn is far more extensive than the ten items in this example. For sampling such masses of data, the program above would be slow and inefficient. Suppose, for instance, a pollster wishes to pick 200 names from an electoral roll of 60,000 voters in a particular constituency, a program such as the one above would need to search the entire list of electors 200 times. To avoid the long wait that this would entail, it would be better to use a single-pass method.

THE SINGLE-PASS SEARCH

The single-pass method would read through the electoral roll once from top to bottom. As each name is considered, a decision is made whether to include that person in the sample. This method can be programmed easily—make the following changes to the last program and reRUN it:

```
50 PRINT AT 0,1; INVERSE 1;
   "□SINGLE-PASS RANDOM
   SAMPLING□"
120 LET a=n: LET c=10
130 FOR j=1 TO 10
140 IF a=0 THEN GOTO 190
150 IF RND*1<=a/c THEN PRINT a$(j):
   GOTO 170
160 LET c=c-1: GOTO 180
170 LET a=a-1: LET c=c-1
```



```
50 PRINT "[graphics] > [graphics] SINGLE —
   PASS "
60 PRINT "[graphics] RANDOM
   SAMPLINGS [graphics]"
120 A=N:C=10
130 FOR J=1 TO 10
140 IF A=0 THEN 190
150 IF RND(1)<=A/C THEN PRINT
   A$(J):GOTO 170
160 C=C−1: GOTO 180
170 A=A−1:C=C−1
```

```
50 MODE1:PRINTTAB(5,3)"SINGLE PASS
   RANDOM SAMPLING"
120 A=N:C=10
130 FOR J=1 TO 10
140 IF A=0 THEN 190
150 IF RND(1)<=A/C THEN
   PRINTA$(J):A=A−1
160 C=C−1
```

Also delete Line 170

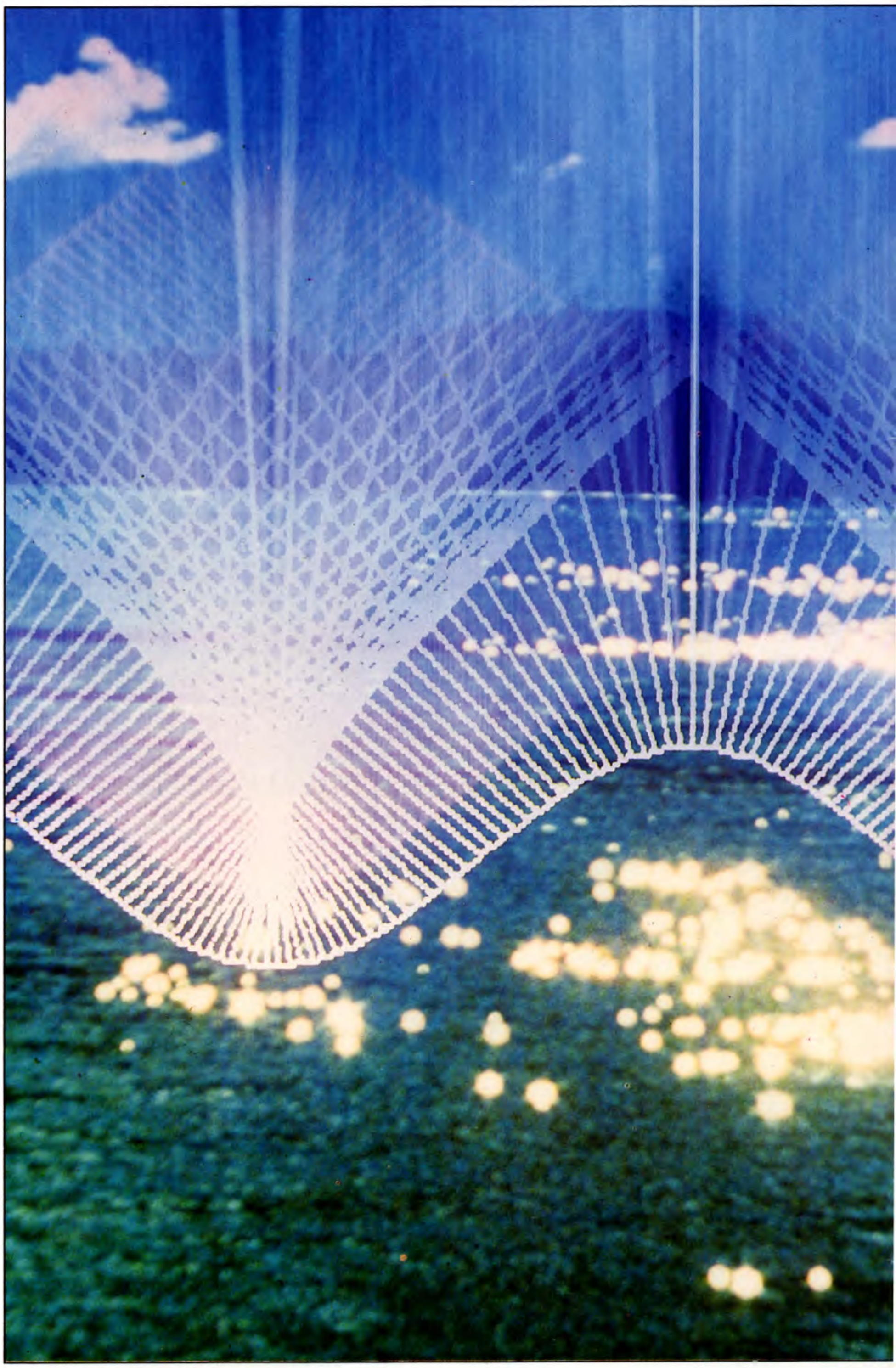
```
50 PRINT@2,"single-pass random
   sampling":PRINT:PRINT
120 A=N:C=10
130 FOR J=1 TO 10
140 IF A=0 THEN 190
150 IF RND(0)<=A/C THEN
   PRINTA$(J):GOTO170
160 C=C−1:GOTO 180
170 A=A−1:C=C−1
```

If now you enter 3, to select three items from the list, the first item (Bonn) is considered first. Should the function RND (Line 150) be less than 3/10, Bonn is selected. Copenhagen is considered next. If Bonn is already in the selection, Copenhagen will have a chance of being selected only if the function RND generates a value of less than 2/9. On the other hand, if Bonn is not already selected, Copenhagen's chance will go up to 3/9. Lines 160 and 170 update the probabilities. When you compare the results of selections from this program with those from the previous one, you should notice that the single-pass method gives samples in alphabetical order.

At first sight, it might appear that with a list of only ten cities, the number of possible samples is small. This is not so. In fact, there are 120 possible different samples of size three, and 252 samples of size five. In a future article, you can see how these ideas can be developed and used in a type of simulation referred to as modelling.

PATTERNS FROM NATURE

A surprising variety of patterns can be produced using very simple graphics routines that superimpose curves or build up patterns of dots. Here are a few ideas to try



The way in which computers can be used to plot the orbit or trajectory of an object falling under gravity was described in the articles on pages 740 to 747 and 797 to 803. Such programs illustrate the simplest aspects of the old science of dynamics. Orbits can, however, generate patterns much richer and more interesting than the parabolas, circles and ellipses in which projectiles and planets move. This article will explain how some of these patterns can be produced by very elementary programming and graphics routines.

Nowadays dynamics is once again a rapidly developing field of research. One reason for this is the realization that there are mathematical ideas underlying dynamics that can be applied much more widely than simply to the motion of bodies acted on by forces. Optical scientists interested in the deviation of starlight by refraction in the atmosphere, industrial chemists studying the progress of a reaction, and biologists concerned with the growth of populations of competing species, all find themselves using the mathematics of dynamics. Another reason is that computers—by enabling simple operations to be repeated many times—have led to the discovery of structural complexity often unsuspected on the basis of the restricted calculations previously possible. A pattern or curve may need to be drawn many, many times before any structure begins to appear. The programs below show how this can be done.

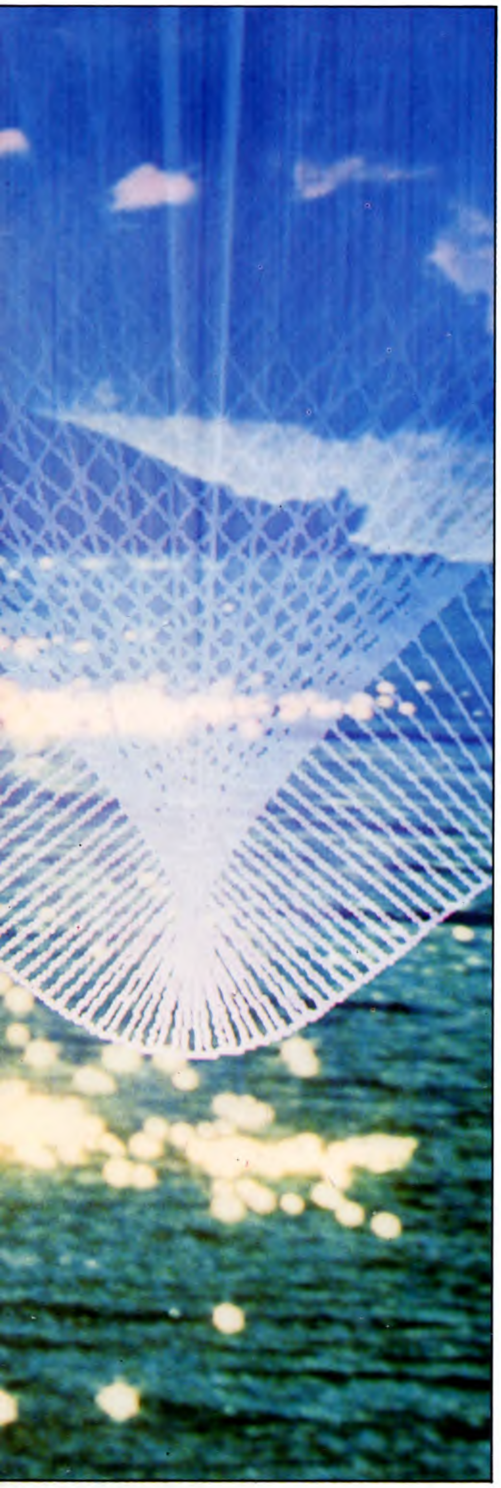
All the programs for the Commodore need a Simons' Basic cartridge or *INPUT*'s hi-res utility. The programs for the Vic need a Super Expander cartridge.

FAMILIES OF ORBITS

When assembled into a collection or *family*, orbits that are individually simple can display striking patterns. To see this for parabolas, enter and RUN the first program.

```
5 LET A$="": FOR N=1 TO 64: LET
   A$=A$+"□": NEXT N
1Ø BRIGHT 1: BORDER Ø: INK 5: PAPER Ø:
   CLS
2Ø LET NMAX=81
3Ø LET DELT=.Ø5
```

- USING PATTERNS IN SCIENCE
- FAMILIES OF ORBITS
- ORBITS AND ENVELOPES
- FOCUSING LINES AT CUSPS
- CATASTROPHE THEORY
- PATTERNS OF DOTS
- DOT CURTAINS AND THE RINGS OF SATURN
- COMPLETE CHAOS
- FISH POPULATIONS



```
40 LET SX=170/SQR 3: LET SY=175
50 FOR N=1 TO NMAX
60 LET A=PI*(-1+2*N/NMAX)
70 PLOT 128,80
80 FOR T=0 TO 3 STEP DELT
90 LET X=T*COS A: LET Y=T*(SIN A-T/2)
100 IF Y<=-.4 THEN GOTO 120
110 LET DX=SX*X+128: LET
   DY=SY*Y+80
111 IF DX<0 OR DX>255 OR DY<0 OR
   DY>175 THEN GOTO 120
115 DRAW DX-PEEK 23677,DY-PEEK 23678
117 PRINT AT 19,0;A$
120 NEXT T
130 NEXT N
```

```
10 HIRES 6,3:COLOUR 6,3
15 BLOCK 0,160,319,199,1
20 NM=81
30 DE=.05
40 SX=160/SQR(3):SY=200
50 FOR N=1 TO NM STEP 2
60 A=π*(-1+2*N/NM)
70 XX=160:YY=100
80 FOR T=0 TO 3 STEP DE
90 X=T*COS(A):Y=T*(SIN(A)-T/2)
100 IFY>-.4THENLINE XX,YY,SX*X+160,
   100-SY*Y,1:XX=SX*X+160:YY=
   100-SY*Y:GOTO110
105 T=3
110 NEXT T,N
130 GOTO 130
```

```
10 GRAPHIC 2:COLOUR 6,1,1,1
15 DRAW 1,0,000 TO 1023,800:
   PAINT 1,0,808
20 NM=81
30 DE=.05
40 SX=512/SQR(3):SY=1000
50 FOR N=1 TO NM STEP 2
60 A=π*(-1+2*N/NM)
70 POINT 0,512,512
80 FOR T=0 TO 3 STEP DE
90 X=T*COS(A):Y=T*(SIN(A)-T/2)
100 IFY>-.4THEN:DRAW 1 TO SX*X+512,
   512-SY*Y:GOTO110
105 T=3
110 NEXT T,N
130 GOTO 130
```

```
10 MODE0
20 NMAX=81
30 DELT=.05
40 SX=800/SQR(3):SY=1300
50 FOR N=1 TO NMAX
60 A=PI*(-1+2*N/NMAX)
70 MOVE 640,341
80 T=0:REPEAT
90X=T*COS(A):Y=T*(SIN(A)-T/2)
100 IF Y>-0.4 THEN DRAW SX*X+640,
   SY*Y+341
110 T=T+DELT:UNTIL Y<=-.4 OR T>3
120 NEXT
```

```
10 PMODE4,1:PCLS1:SCREEN1,0
20 NM=81:PI=4*ATN(1)
30 DE=.05
40 SX=160/SQR(3):SY=230
50 FOR N=1 TO NM
60 A=PI*(-1+2*N/NM)
70 DRAW"BM127,118"
80 FOR T=0 TO 3 STEP DE
90 X=T*COS(A):Y=T*(SIN(A)-T/2)
100 IF Y>-.4 THEN LINE-(SX*X+127,
   118-SY*Y),PRESET ELSE T=3
110 NEXT T
120 NEXT N
130 GOTO 130
```

This program simulates the paths of falling drops of water sprayed from the head of a garden sprinkler, or, in a more modern application, neutrons squirted from a thin pipe connected to a reactor. In this case the family of orbits consists of all the parabolic paths that emerge from a particular point in different directions but with the same speed. The pattern formed by this family is the outer curve which each orbit touches. This curve, called the *envelope* of the family, happens, itself, to be a parabola in this case (in gunnery it is called the 'bounding parabola' because it is the boundary of the region that can be reached by projectiles of a fixed initial speed). It is important to realize that the envelope is a property of the whole family of parabolic orbits and has no meaning for any single one of them. Thus envelopes are perfect ill-ustrations of the fact that the whole can be

greater than the sum of its parts.

In the program, the equation for the orbits appears on Line 9Ø. X is horizontal distance, Y is vertical distance, T is time (starting from zero at the instant of emission), and each orbit in the family is labelled by A, which is an angle giving the direction in which it starts out. There are NMAX or NM such directions; try changing the value of NMAX or NM in Line 2Ø.

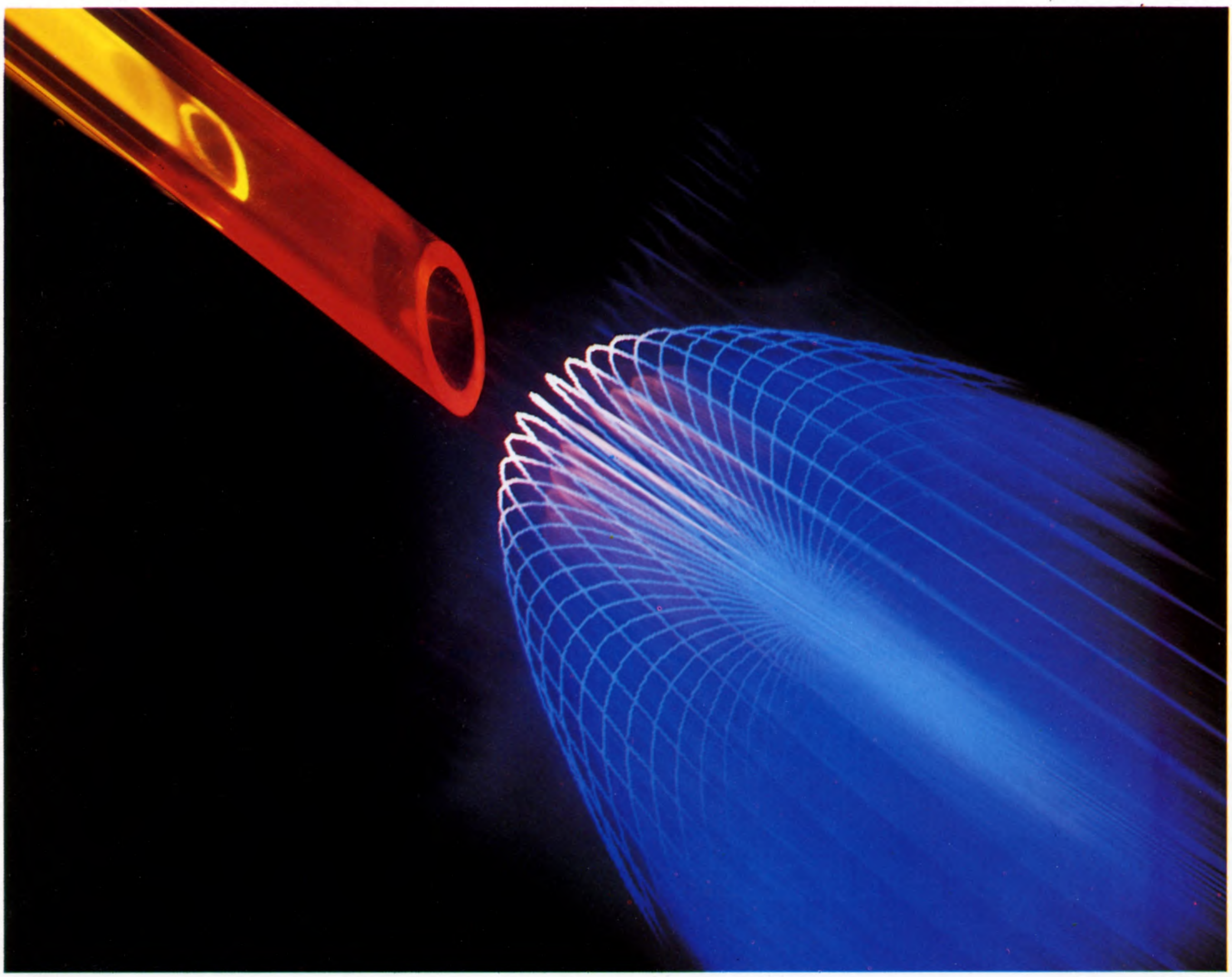
FOCUSING

Even straight lines can form families with interesting envelopes, as the second program shows.

```
1Ø BORDER Ø: INK 7: PAPER Ø: CLS:
   LET N=2
3Ø LET YØ=8Ø/N/N
4Ø PLOT Ø,-6-YØ*2
5Ø FOR X=Ø TO 255 STEP 2
6Ø LET Y=169-YØ-YØ*COS (N*2*PI*X/255)
7Ø DRAW X-PEEK 23677,175-Y-PEEK
   23678
8Ø LET XT=X+2*YØ*N*PI*Y/255*SIN (N*2*
   PI*X/255)
85 LET Y1=Ø: IF XT<Ø THEN LET XT=Ø:
   LET Y1=Y+255*X/(2*N*PI*YØ*SIN (N*
   2*PI*X/255))
86 IF XT>255 THEN LET XT=255: LET
   Y1=Y-(255-X)*255/(2*N*PI*YØ*SIN
   (N*2*PI*X/255))
9Ø DRAW XT-PEEK 23677,175-Y1-PEEK
   23678
1ØØ PLOT X,175-Y
11Ø NEXT X
12Ø GOTO 12Ø
```

```
1Ø HIRES 6,3:COLOUR 6,3
2Ø N=2
3Ø YØ=8Ø/N/N
4Ø XX=Ø:YY=INT(16Ø-YØ*2)
5Ø FOR X=Ø TO 319 STEP 2
6Ø Y=16Ø-YØ-YØ*COS(N*2*π*X/319)
7Ø LINE XX,YY,X,Y,1:XX=X:YY=Y
8Ø XT=X+2*YØ*N*π*Y/319*SIN(N*2*π*
   X/319)
85 Y1=Ø:IF XT<Ø THEN XT=Ø:Y1=Y+
   199*X/(2*N*π*YØ*SIN(N*2*π*X/199))
86 IF XT>319 THEN XT=319:Y1=Y-
   (199-X)*199/(2*N*π*YØ*SIN(N*2*π*
   X/199))
9Ø LINE XX,YY,XT,Y1,1
11Ø NEXT X
12Ø GOTO 12Ø
```

```
1Ø GRAPHIC 2:COLOR 6,1,1,1
2Ø N=2
3Ø YØ=2ØØ/N/N
4Ø XX=Ø:YY=INT(512-YØ*2)
5Ø FOR X=Ø TO 1Ø23 STEP 2Ø
6Ø Y=512-YØ-YØ*COS(N*2*π*X/1Ø23)
7Ø DRAW 1,XX,YYTO X,Y:XX=X:YY=Y
9Ø XT=X+2*YØ*N*π*Y/1Ø23*SIN(N*2*π*
```



```
   X/1Ø23)
85 Y1=Ø:IF XT<Ø THEN XT=Ø:Y1=Y+
   1Ø23*X/(2*N*π*YØ*SIN(N*2*π*X/1Ø23))
86 IF XT>1Ø23 THEN XT=1Ø23:Y1=Y−
   (1Ø23−X)*1Ø23/(2*N*π*YØ*SIN(N*2*π*
   X/1Ø23))
9Ø DRAW 1,XX,YYTO XT,Y1
11Ø NEXT X
12Ø GOTO 12Ø
```

```
1Ø MODEØ
2Ø N=2
3Ø YØ=4ØØ/N/N
4Ø MOVEØ,YØ*2+3Ø
5Ø FOR X=Ø TO 1283 STEP 9
6Ø Y=YØ+3Ø+YØ*COS(N*2*PI*X/1279)
7Ø DRAW X,Y
8Ø XTOP=X+2*YØ*N*PI*(1Ø23−Y)/1279*
   SIN(N*2*PI*X/1279)
9Ø DRAW XTOP,1Ø23
1ØØ MOVEX,Y
11Ø NEXT
```

```
1Ø PMODE4,1:PCLS1:SCREEN1,Ø
2Ø N=2:PI=4*ATN(1)
3Ø YØ=8Ø/N/N
4Ø DRAW"BMØ,"+STR$(INT(186−YØ*2))
5Ø FOR X=Ø TO 255 STEP 2
6Ø Y=186−YØ−YØ*COS(N*2*PI*X/255)
7Ø LINE−(X,Y),PSET
8Ø XT=X+2*YØ*N*PI*Y/255*SIN(N*2*PI*X/
   255)
85 Y1=Ø:IF XT<Ø THEN XT=Ø:YZ=Y+
   255*X/(2*N*PI*YØ*SIN(N*2*PI*X/255))
86 IF XT>255 THEN XT=255:Y1=Y−
   (255−X)*255/(2*N*PI*YØ*SIN(N*2*PI*X/
   255))
9Ø LINE−(XT,Y1),PRESET
1ØØ DRAW"BM"+STR$(X)+","+STR$
   (INT(Y))
11Ø NEXT
12Ø GOTO12Ø
```

This program simulates light rays bent by refraction through a wavy, curved surface, for example sunlight refracted by ripples on water in a swimming pool. The family of orbits consists of all the straight lines at right angles to a wavy curve of sine form. For light rays the envelope is the curve corresponding to *focusing*. Focusing is particularly intense near the troughs (minima) of the sine wave, where the envelope has sharp points called *cusps*. These cusps are the bright points of light you see amongst the ripples. The existence of cusps is predicted by the recently developed mathematics of envelopes, called *catastrophe theory* (whose dramatic name originated in applications of the same mathematics to the collapse of bridges and the capsizing of ships).

In the program, the wavy initial curve is specified in Line 6Ø and the rays that start out from it are defined in Line 8Ø. The number of troughs of the wavy curve is N; try changing the value of N in Line 2Ø (N=1 is especially recommended).

Trajectories that are themselves wavy sine curves are assembled into a family in the third program.

```
1Ø BORDER Ø: PAPER Ø: INK 7: CLS
2Ø LET QM=SQR (2*LN (3))
3Ø FOR Q=−QM TO QM*1.ØØ1 STEP QM/3Ø
4Ø PLOT Ø,75+Q*75/QM
5Ø FOR T=Ø TO 3*PI STEP .2
6Ø LET X=Q*COS (EXP (−Q*Q/2)*T)
7Ø DRAW (T*24Ø/3/PI)−PEEK
   23677,75+(X*75/QM)−PEEK 23678
8Ø NEXT T
9Ø NEXT Q
```

```
1Ø HIRES Ø,1:COLOUR 1,6:MULTI 4,3,7
2Ø QM=SQR(2*LOG(3))
3Ø FOR Q=−QM TO QM*1.ØØ1 STEP
   QM/3Ø
4Ø XX=Ø:YY=INT(1ØØ−Q*9Ø/QM)
5Ø FOR T=Ø TO 3*π STEP .2
6Ø X=Q*COS(EXP(−Q*Q/2)*T)
7Ø LINE XX,YY,T*17/3*π,1ØØ−X*9Ø/QM,RND
   (1)*3+1
75 XX=T*17/3*π:YY=1ØØ−X*9Ø/QM
8Ø NEXT T,Q
9Ø GOTO 9Ø
```

```
1Ø GRAPHIC 1:COLOR 6,1,3,7
2Ø QM=SQR(2*LOG(3))
3Ø FOR Q=−QM TO QM*1.ØØ1 STEP
   QM/3Ø
4Ø POINT Ø,Ø,INT(512−Q*5ØØ/QM)
5Ø FOR T=Ø TO 3*π STEP .6
6Ø X=Q*COS(EXP(−Q*Q/2)*T)
7Ø DRAW RND(1)*3+1 TO T*116/ 3*π,
   512− X* 5ØØ/ QM
8Ø NEXT T,Q
9Ø GOTO 9Ø
```

```
1Ø MODEØ
2Ø QM=SQR(2*LN(3))
3Ø FOR Q=−QM TO QM*1.ØØ1 STEP QM/3Ø
4Ø MOVE Ø,Q*45Ø/QM+5ØØ
5Ø FOR T=Ø TO 3*PI STEP .2
6Ø X=Q*COS(EXP(−Q*Q/2)*T)
7Ø DRAW T*12ØØ/3/PI,X*45Ø/QM+5ØØ
8Ø NEXT
9Ø NEXT
```

```
1Ø PMODE4,1:PCLS1:SCREEN1,Ø
2Ø QM=SQR(2*LOG(3)):PI=4*ATN(1)
3Ø FOR Q=−QM TO QM*1.ØØ1 STEP
   QM/3Ø
4Ø DRAW"BMØ,"+STR$(INT(1ØØ−Q*9Ø/
   QM))
5Ø FOR T=Ø TO 3*PI STEP .2
6Ø X=Q*COS(EXP(−Q*Q/2)*T)
7Ø LINE−(T*24Ø/3/PI,1ØØ−X*9Ø/QM),
   PRESET
8Ø NEXT T
9Ø NEXT Q
1ØØ GOTO 1ØØ
```

This program simulates light rays passing along a thin glass optical fibre whose refractive index varies across its width, or (on a much smaller scale) electrons winding between planes of atoms in a crystal placed in the beam of an electron microscope. The family consists of orbits starting out parallel to each other at the left-hand edge of the screen; each orbit undulates about the central line at a rate that depends on its starting point. Although this family is very different from the last program, the envelope curves also display intense focusing at cusp catastrophe points.

In the program, orbits are specified in Line 6Ø by giving their distance X from the centre line at time T. Different orbits in the family are labelled by Q. There are 30 orbits in the program as written; to get more or fewer, change the number at the end of Line 3Ø.

DOT PATTERNS

Assembling orbits into families is not the only method of obtaining interesting patterns. Another way is to follow the orbits for *very long times*, with the result that considerable complexity can develop even when the mathematics of the orbit is relatively simple. In displaying these orbits it is not usually advisable to plot the continuous curve giving the position at every instant, because over long times this would just fill the screen with a mess like tangled wool. Instead the orbit is plotted at regular intervals (once every second, for example), as though the trajectory were viewed in the flashing light of a stroboscope. Examples of the resulting dot patterns are given in the next three programs. The position of dots on the screen does not always correspond to the real, spatial position of physical objects whose motion the programs simulate; sometimes it is an abstract representation, in which horizontal screen position corresponds to position and vertical screen position to speed.

The fourth program will take several minutes to RUN.

```
10 BORDER Ø: PAPER Ø: INK 7: CLS
30 LET A=76.11
40 LET ALF=A*PI/18Ø: LET C=COS (ALF)
50 LET S=SIN (ALF)
60 LET NMAX=2ØØ
70 LET M=52
80 FOR J=1 TO M
90 LET X=Ø: LET Y=J/M
100 FOR N=1 TO NMAX
110 LET W=X
120 LET X=X*C-(Y-X*X)*S: LET
   Y=W*S+(Y-W*W)*C
130 IF ABS (X)>4 OR ABS (Y)>4 THEN
   GOTO 86Ø
135 IF X>1 OR Y>1 THEN GOTO 15Ø
140 PLOT X*128+128,Y*85+85
150 NEXT N
160 NEXT J
```

```
1Ø HIRES Ø,1:COLOUR 1,6:MULTI 5,3,7
2Ø A=76.1
3Ø AL=A*π/18Ø:C=COS(AL)
4Ø S=SIN(AL)
5Ø NM=2ØØ
6Ø M=52
7Ø FOR J=1 TO M
8Ø X=Ø:Y=J/M:CL=RND(1)*3+1
9Ø FOR N=1 TO NM
1ØØ W=X
11Ø X=X*C-(Y-X*X)*S:Y=W*S+(Y-
   W*W)*C
12Ø IF ABS(X)>4 OR ABS(Y)>4 THEN 16Ø
125 IF ABS(Y)>1 OR ABS(X)>1 THEN 14Ø
13Ø PLOT 8Ø+X*79,1ØØ-Y*99,CL
14Ø NEXT N,J
15Ø GOTO 15Ø
```

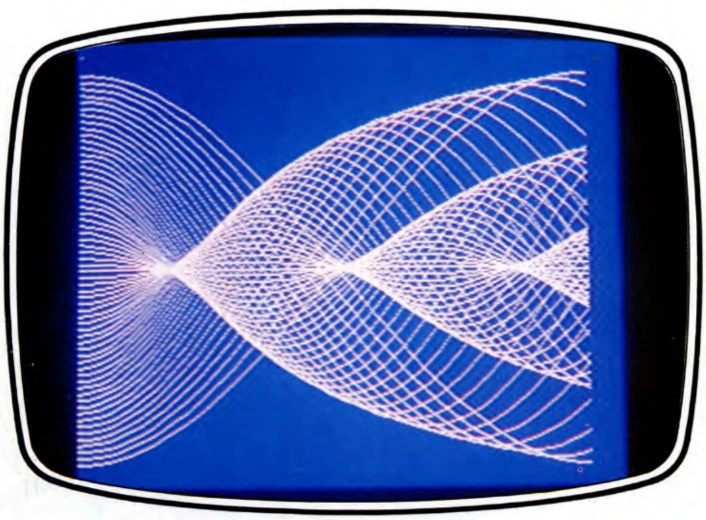
```
1Ø GRAPHIC 1:COLOR 6,1,3,7
2Ø A=76.1
3Ø AL=A*π/18Ø:C=COS(AL)
4Ø S=SIN(AL)
5Ø NM=2ØØ
6Ø M=52
7Ø FOR J=1 TO M
8Ø X=Ø:Y=J/M:CL=RND(1)*3+1
9Ø FOR N=1 TO NM
1ØØ W=X
11Ø X=X*C-(Y-X*X)*S:Y=W*S+
   (Y-W*W)*C
12Ø IF ABS(X)>4 OR ABS(Y)>4 THEN 16Ø
125 IF ABS(Y)>1 OR ABS(X)>1 THEN 14Ø
13Ø POINT CL,512+X*512,512-Y*512
14Ø NEXT N,J
15Ø GOTO 15Ø
```

```
1Ø MODEØ
2Ø A=76.11
3Ø ALF=A*PI/18Ø:C=COS(ALF)
4Ø S=SIN(ALF)
5Ø NMAX=2ØØ
6Ø M=52
7Ø FOR J=1 TO M
8Ø X=Ø:Y=J/M
9Ø FOR N=1 TO NMAX
1ØØ W=X
11Ø X=X*C-(Y-X*X)*S:Y=W*S+
   (Y-W*W)*C
12Ø IF ABS(X)>4 OR ABS(Y)>4 THEN END
13Ø PLOT 69,X*64Ø+64Ø,Y*512+512
14Ø NEXT
15Ø NEXT
```

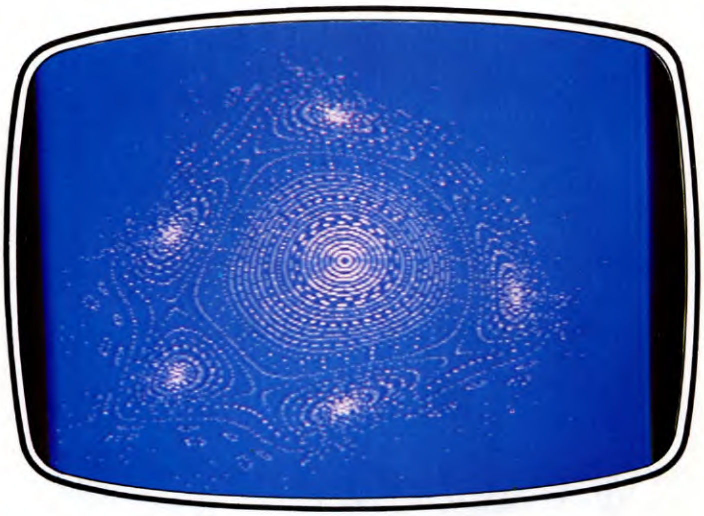
```
1Ø PMODE4,1:PCLS1:SCREEN1,Ø
2Ø A=76.11:PI=4*ATN(1)
3Ø AL=A*PI/18Ø:C=COS(AL)
4Ø S=SIN(AL)
5Ø NM=2ØØ
6Ø M=52
7Ø FOR J=1TOM
8Ø X=Ø:Y=J/M
9Ø FORN=1TONM
1ØØ W=X
11Ø X=X*C-(Y-X*X)*S:Y=W*S+
   (Y-W*W)*C
12Ø IF ABS(X)>4 OR ABS(Y)>4 THEN 16Ø
125 IF ABS(Y)>1 OR ABS(X)>1 THEN14Ø
13Ø PRESET(128+X*128,96-Y*96)
14Ø NEXT N
15Ø NEXT J
16Ø GOTO16Ø
```

This program simulates motion of subatomic particles (such as protons or electrons) in an accelerator, or the windings of a line of force in the magnetic bottle of a fusion power device. Mathematically, what the program does is to take a series of initial points and move them about the screen by repeating a rule that says: 'rotate about the middle of the screen by a fixed angle, apart from a slight modification'. Without the 'slight modification' the orbits of the points would all be circles, and indeed those near the centre are roughly circles. But the modification has a dramatic effect on points initially far from the centre: their orbits may be a series of 'islands', or they may escape from the screen altogether. More refined computer graphics, using high magnification, reveals tiny islands everywhere, distributed amongst the large ones.

In the program, the number of initial points is M, specified in Line 6Ø; if you get tired of waiting for the picture you can reduce this number. Each of these points is plotted for NMAX or NM repetitions of the transform-



Focusing light in a glass fibre



Islands of dots or lines of force

ation rule; NMAX or NM is specified in Line 5Ø. The rule itself is contained in Lines 1ØØ and 11Ø, which describe how the horizontal and vertical screen coordinates X and Y are altered at each repetition. The angle of the unmodified rotation, in degrees, is A, specified in Line 2Ø; you should experiment with different values of A (try 9Ø).

An unexpected pattern made of dots generated by a single initial point is produced by the next program which you should now enter and run:

```
20 LET K=51.3: BORDER Ø: INK 7: PAPER Ø:
   CLS
3Ø LET X=1/PI: LET P=Ø
4Ø LET A=1/SQR 5
5Ø FOR N=1 TO 1ØØØØ
6Ø LET Y=X-.5: LET X=X+A-INT (X+A)
7Ø LET P=P-Y
8Ø PLOT X*255,P*K+85
9Ø NEXT N
```

```
1Ø HIRES Ø,1:COLOUR 1,6:
   MULTI 5,3,7:K=6Ø
2Ø X=1/π:P=Ø
3Ø A=1/SQR(5)
4Ø FOR N=1 TO 1ØØØØ
5Ø Y=X-.5:X=X+A-INT(X+A)
6Ø P=P-Y
7Ø PLOT X*159,1ØØ-P*K,RND
   (1)*3+1
8Ø NEXT N
9Ø GOTO 9Ø
```

```
1Ø GRAPHIC 1:COLOR 6,1,3,7:
   K=3ØØ
2Ø X=1/π:P=Ø
3Ø A=1/SQR(5)
4Ø FOR N=1 TO 1ØØØØ
5Ø Y=X-.5:X=X+A-INT(X+A)
6Ø P=P-Y
7Ø POINT RND(1)*3+1,X*1Ø23,
   512—P*K
8Ø NEXT N
9Ø GOTO 9Ø
```

```
1Ø MODEØ:K=3ØØ
2Ø X=1/PI:P=Ø
3Ø A=1/SQR(5)
4Ø FOR N=1 TO 1ØØØØ
5Ø Y=X-.5:X=X+A-INT(X+A)
6Ø P=P-Y
7Ø PLOT 69,X*1279,P*K+512
8Ø NEXT
```

```
1Ø PMODE4,1:PCLS1:SCREEN1,Ø:
   K=6Ø
2Ø X=1/(4*ATN(1)):P=Ø
3Ø A=1/SQR(5)
4Ø FOR N=1 TO 1ØØØØ
5Ø Y=X-.5:X=X+A-INT(X+A)
6Ø P=P-Y
7Ø PRESET(X*255,128-P*K)
8Ø NEXT N
9Ø GOTO9Ø
```

This program is an abstract simulation of *resonance*, where the frequencies of two physical effects may clash. For example, one frequency might be that of an asteroid's motion round the sun, and the other might be the frequency with which the asteroid is disturbed by the gravitational pull of the planet Jupiter. In this case an important question is: do the disturbing pulls mount up and throw the asteroid out of its orbit, or do they force it into a stable orbit? The answer is that this depends on the *ratio* of the two frequencies (that is, on one divided by the other). The particles in Saturn's rings are affected by similar forces.

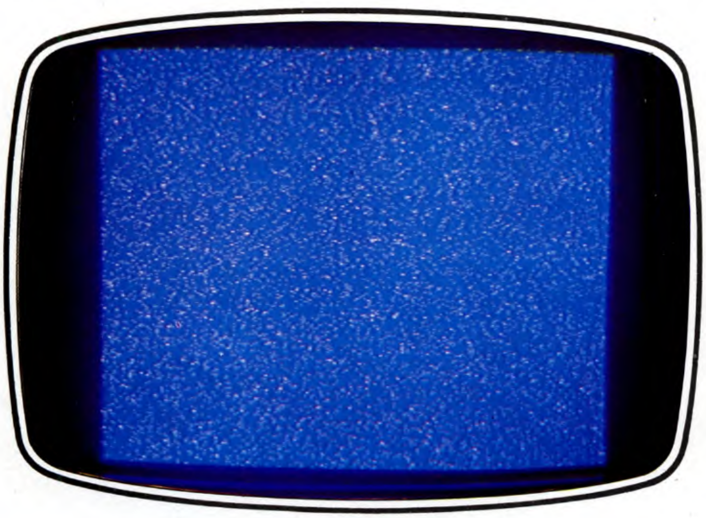
In the program, the ratio is called A and its value (which must be less than 1) is specified in Line 3Ø. The number of repetitions of the resonance transformation is 10000 and is specified at the end of Line 4Ø; reduce this value if you get tired of waiting. The transformation itself is on Lines 5Ø and 6Ø, and is the rule for changing the horizontal (X) and vertical (P) positions of the dot on the screen.

As written, the delicate curtain pattern is produced by A=1/SQR(5) which equals 0.4472136. This is not the ratio of two whole numbers (it is an 'irrational number') and corresponds to *nonresonance*. To get resonance you should try a number that is the ratio of two whole numbers, such as A=9/2Ø which equals 0.45. Then try a different irrational number, such as A=1/PI. (If you want to reduce the vertical scale to display more of the picture, diminish the value of K.

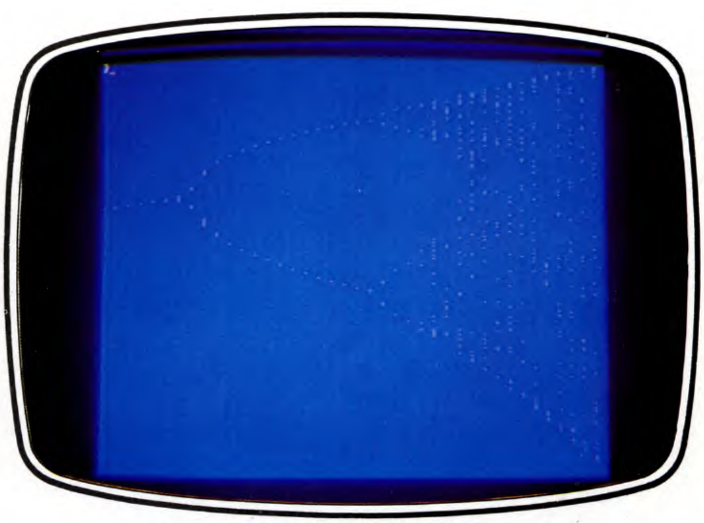
CHAOS

It is a remarkable fact that some simple rules produce *no pattern at all*. Such an example is the program below which you should enter and RUN:

```
1Ø BORDER Ø: PAPER Ø: INK 7: CLS
2Ø LET X=1/PI
3Ø LET Y=1/PI
4Ø FOR M=1 TO 1ØØØØ
5Ø LET X=X+Y-INT (X+Y)
6Ø LET Y=X+Y-INT (X+Y)
7Ø PLOT X*255,Y*175
8Ø NEXT M
```



Some simple rules produce chaos



Population changes in a fish pond

```
10 HIRES Ø,1:COLOUR 6,6:
   MULTI 5,3,7
20 X=1/π
30 Y=1/π
40 FOR M=1 TO 10000
50 X=X+Y-INT(X+Y)
60 Y=X+Y-INT(X+Y)
70 PLOT X*159,199-Y*199,
   RND(1)*3+1
90 NEXT M
90 GOTO 90
```

```
10 GRAPHIC 1:COLOR 6,1,3,7
20 X=1/π
30 Y=1/π
40 FOR M=1 TO 10000
50 X=X+Y-INT(X+Y)
60 Y=X+Y-INT(X+Y)
70 POINT RND(1)*3+1,X*1023,
   1023-Y*1023
80 NEXT M
90 GOTO 90
```

```
10 MODEØ
20 X=1/PI
30 Y=1/PI
40 FOR M=1 TO 10000
50 X=X+Y-INT(X+Y)
60 Y=X+Y-INT(X+Y)
70 PLOT69,X*1279,Y*1023
80 NEXT
```

```
10 PMODE4,1:PCLS1:SCREEN1,Ø
20 PI=4*ATN(1):X=1/PI
30 Y=1/PI
40 FORM=1TO10000
50 X=X+Y-INT(X+Y)
60 Y=X+Y-INT(X+Y)
70 PRESET(X*255,192-Y*191)
80 NEXT M
90 GOTO 90
```

This program simulates the erratic bouncing of metal spheres in a pinball machine, or the motion of molecules in a gas, or of a roulette wheel, or indeed any dynamic system whose orbits are so unpredictable as to be indistinguishable from what would be generated by purely random processes. And yet the ten thousand points on the screen are not the result of random sprinkling but are generated by the program from a single point by repeating a rule that is purely deterministic—that is, it contains no random element.

The rule is based on regarding the screen as the 'unit square' on which horizontal distance X and vertical distance Y range from zero to one, and operates in three stages, constituting Lines 5Ø and 6Ø of the program. First, the X and Y coordinates of a point are added to produce a new X; second, this new X is added to Y to produce a new Y; third, if either of the new X or Y lies outside the range 0 to 1, an appropriate whole number is subtracted in order to bring the transformed point back into the unit square (in the program this subtraction is implemented by the INT function in Lines 5Ø and 6Ø).

FISH POPULATIONS

How does the population of fish in a pond change over many generations? This depends on the rule that determines how the population changes from one generation to the next. Such a rule must incorporate both the tendency of the population to increase because each set of parents produces more than two offspring, and the tendency of the population to decrease when it gets so large that the finite food supply in the pond cannot sustain it. Depending on the precise balance between these two dependencies a fish population may settle down to a stable fixed value, or alternate regularly between two or more values, or change apparently randomly between successive generations.

The final program employs graphics and sound to illustrate these different possibilities. Horizontal screen position, denoted by A, corresponds to the balance between breeding and food supply and hence to the rule relating successive generations. Vertical screen position, denoted by Y, corresponds to the population, represented by a point jumping up or down at each generation. The population Y is plotted on the screen only when it has settled down to its stable value or set of alternating values, called the *attractor set*. But the way in which the population homes in on the attractor can be heard because the program makes the computer emit a sound whose pitch (frequency) is proportional to the population.

```
10 BORDER Ø: INK 7: PAPER Ø: CLS
20 LET S=.03 LET NMIN=50: LET
   NMAX=80
30 FOR A=2.8 TO 4 STEP S
40 LET Y=1/PI
50 FOR N=1 TO NMAX
60 LET Y=A*Y*(1-Y)
70 IF N>NMIN THEN PLOT 255*
   (A-2.8)/1.2,Y*175
80 BEEP .0075,Y*20
90 NEXT N
100 NEXT A
```

```
10 HIRES Ø,1:COLOUR 1,6:
   MULTI 5,3,7
15 POKE 54296,15:POKE 54277,64
20 S=1/160:NN=50:NX=80
30 FOR A=2.8 TO 4 STEP S
40 Y=.25/ATN(1)
45 POKE 54276,33
50 FOR N=1 TO NX
60 Y=A*Y*(1-Y)
70 IF N>NN THEN PLOT 159*(A-2.8)/1.2,
   199-Y*199,RND(1)*3+1
80 POKE 54273,1+255*Y
90 NEXT N
95 POKE 54276,32
100 NEXT A
110 GOTO 110
```

```
10 GRAPHIC 1:COLOR 6,1,3,7
15 POKE 36878,15
20 S=.01:NN=50:NX=80
30 FOR A=2.8 TO 4 STEP S
40 Y=.25/ATN(1)
50 FOR N=1 TO NX
60 Y=A*Y*(1-Y)
70 IF N>NN THEN:POINT RND(1)*3+1,
   1023*(A-2.8)/1.2,1023-Y*1023
80 POKE 36876,128+127*Y
90 NEXT N
95 POKE 36876,Ø
100 NEXT A
110 GOTO 110
```

```
10 MODEØ
20 S=.03:NMIN=50:NMAX=80
30 FOR A=2.8 TO 4 STEP S
40 Y=1/PI
50 FOR N=1 TO NMAX
60 Y=A*Y*(1-Y)
70 IF N>NMIN THEN PLOT69,1279*
   (A-2.8)/1.2,Y*1023
80 SOUND1,-15,255*Y,1
90 NEXT
100 NEXT
```

```
10 PMODE4,1:PCLS1:SCREEN1,Ø
20 S=1/127:NN=50:NX=80
30 FORA=2.8 TO4 STEP S
40 Y=.25/ATN(1)
50 FORN=1TONX
60 Y=A*Y*(1-Y)
70 IF N>NN THENPRESET(255*(A-2.8)/1.2,
   192-Y*191)
80 SOUND1+255*Y,1
90 NEXT N
100 NEXT A
110 GOTO110
```



It is clear that the attractor undergoes drastic changes as the amount of food is slowly altered. To start with (on the left of the screen), the attractor is a single point, indicating an eventually stable fish population. This means there is enough food for the fish to live healthily and breed well. If the population increases too much there will be less food to go around so some fish will die of starvation. This leaves more food for the survivors so they grow and breed well again. Eventually the numbers in each generation settle down to a constant stable population. In the program you'll hear the sound oscillating between each generation but settling down to a stable value each time.

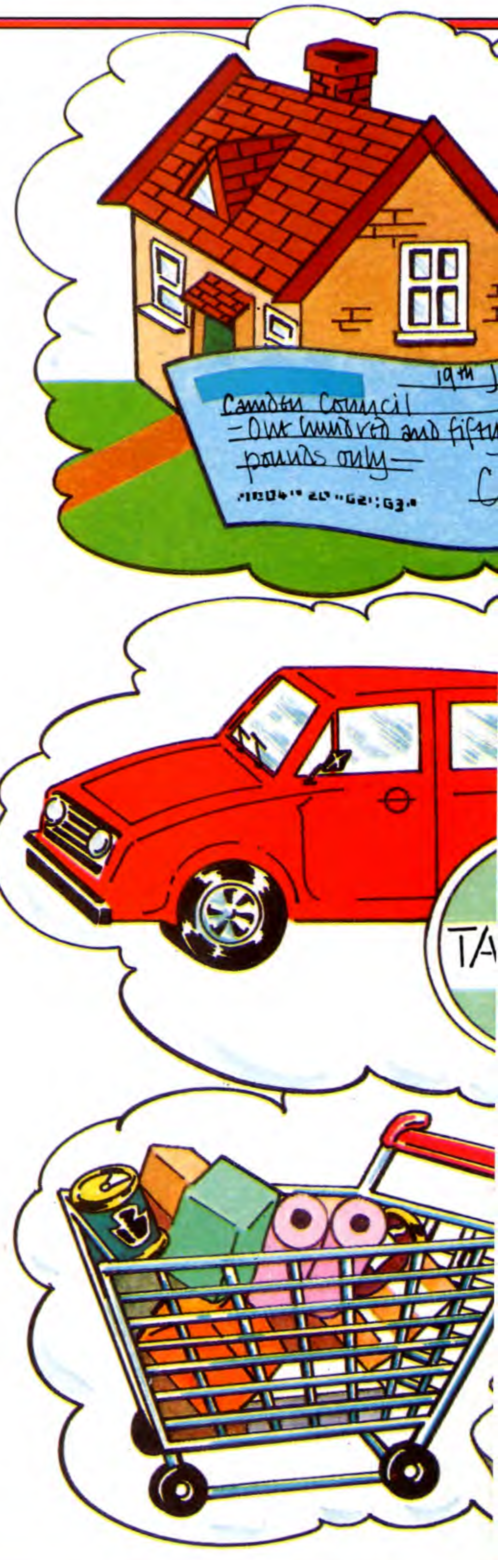
Suddenly, at a certain value of A, as the food is increased past a certain level, the attractor is seen and heard to branch into two or *bifurcate*, indicating a population eventually alternating between two values. Later, with even more food, the attractor bifurcates again, indicating four alternating population values. More and more divisions occur, forming an infinite sequence of which only the first few are resolved by this program. The successive bifurcations accumulate at a finite A value, corresponding to a fish population that fluctuates among infinitely many values without ever settling down.

Much excitement has been generated by the discovery that this 'bifurcation tree', ending in chaos, arises in a wide range of mathematical contexts. An important application is to the way in which the motion of a flowing liquid changes in stages from smooth (stable attractor) to turbulent (chaotic attractor) as the speed increases (from the lazy river to the raging torrent). The same thing can be seen as smoke from a cigarette rises gently then suddenly spirals and swirls into turbulent eddies.

In this program the population evolution rule is specified in Line 6Ø. Use of the sound command gives a vivid impression of how the orbit reaches the attractor, but does slow the program down. To get a clearer picture of the attractor itself in a reasonable time first delete Line 8Ø. You can increase the resolution by changing Line 2Ø so that S = Ø.ØØ5, NMIN or NN = 2ØØ and NMAX or NX = 3ØØ.

TAILORING SPREADSHEETS

Find out where your money goes or plan out the future of your business with this handy spreadsheet program. Add some more lines to the program started last time



Since a spreadsheet starts out as a blank sheet of paper (or a blank screen!) it is sometimes difficult to know exactly what to use it for. The examples given last time and the ideas below should help, and you should be able to tailor one of them to your own needs. And of course, there is no need to limit yourself to just one sheet. The program can be used to set up any number of spreadsheets which can be saved and reloaded at any time.

Here are a few examples. You could have one sheet to record and plan out your house expenses, where entries would go under headings such as rates, mortgage, insurance, repairs, etc. But if you were making a lot of repairs you might want to set up a separate sheet for these. You could itemise the different types of repairs, as well as decorations and furnishings for individual rooms, showing how much was spent each month or each quarter. The program could then total the expenditure for each type of repair over the whole house, or the total expenditure for each room.

Yet another sheet could include all general family expenses such as food and drink, clothing, car, holidays and entertainment. It could list these under different weeks or months, or under different members of the family.

But because the sheet starts off completely blank you can use it for any information that needs to be laid out in a logical way. For example a club membership spreadsheet could list members' names, numbers and subscriptions paid, as well as listing and adding up the attendances each week—using labels for most of the entries.

A spreadsheet is also ideal for entering details of any survey. In fact if you look back at the article on pages 269 to 275 you'll see that the spreadsheet is really just a much more sophisticated version of a two-dimensional array. The spreadsheet allows you much more control over the way the entries are made and, because you can enter equations as well, the result of the survey can be calculated immediately.

Business uses are more obvious, and are virtually unlimited. Spreadsheets can hold details of invoices, displaying details of the items along with their cost, VAT, discounts and so on. They can work out staff salaries, where the names of the staff, hours worked, rate of pay, allowances and tax are all listed or worked out by the spreadsheet. They can be used for stocktaking, for general accounts and for budgeting—including making alternative forecasts for the future at the push of a button.

ENTERING THE PROGRAM

The section of program given here joins on to the one given last time. The remainder appears in the next article which also gives detailed instructions on how to use it. So LOAD in the last part, add these lines and SAVE it ready for next time when you will be able to RUN the complete program.

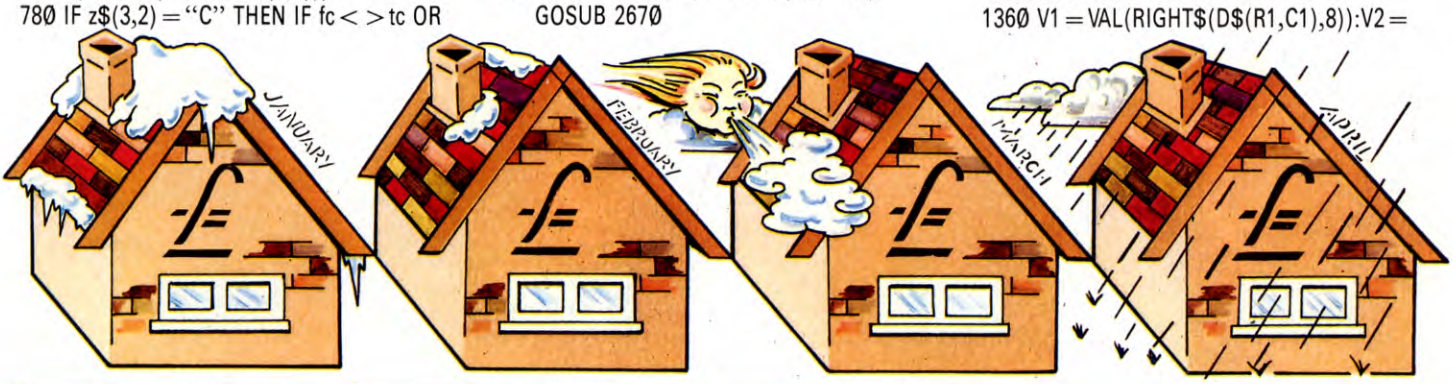
```
420 FOR a=fc TO tc: FOR b=fr TO tr
430 IF z$(3,2)="C" THEN LET
   v(2)=v(2)+1: LET
   v(4)=v(4)+(v(3)<>26)
440 IF z$(3,2)="R" THEN LET
   v(3)=v(3)+(v(3)<>26): LET
   v(1)=v(1)+1
450 IF v(1)<25 AND v(2)<31 AND
   v(3)=26 THEN GOTO 470
460 IF v(1)>24 OR v(2)>30 OR v(3)>24
   OR v(4)>30 THEN GOTO 570
470 IF v(1)<1 OR v(2)<1 OR v(3)<1 OR
   v(4)<1 THEN GOTO 570
480 LET a$=CHR$ (v(1)+64)+STR$
   v(2)+CHR$ (v(3)+64)+STR$ v(4)
   +o$
485 LET c=LEN a$: IFC>8THEN RETURN
490 RESTORE 1630: FOR q=1 TO 11: LET
   f=0: READ m$: FOR w=1 TO c
500 IF m$(w)="A" THEN GOSUB 1650: IF f
   THEN GOTO 560
510 IF m$(w)="N" THEN GOSUB 1670: IF f
   THEN GOTO 560
520 IF m$(w)="Z" THEN GOSUB 1710: IF f
   THEN GOTO 560
530 IF m$(w)="O" THEN GOSUB 1690: IF f
   THEN GOTO 560
540 NEXT w: LET z=q: GOSUB 1140:
   IF NOT f THEN LET s$="□□□
   □□□□□": FOR w=1 TO c: LET
   s$(w)=a$(w): NEXT w: LET d$(b,a,9 TO
   16)=s$: LET d$(b,a,18)=CHR$ z: LET
```

- PLANNING THE SPREADSHEET
- GENERAL USES
- FAMILY BUDGET
- HOUSEHOLD EXPENSES
- CLUB MEMBERSHIP DETAILS
- BUSINESS USES
- INVOICES
- STAFF SALARIES
- STOCKTAKING
- ENTERING THE PROGRAM



```
    d$(b,a,17)="1": NEXT b: NEXT a:
    RETURN
550 GOTO 570
560 NEXT q
570 RETURN
580 LET e=c: LET a$="□□□"
590 PRINT #1;AT Ø,x; BRIGHT 1;"□"
600 PAUSE Ø: LET i=CODE INKEY$
610 IF i>88 THEN GOTO 600
620 IF i=13 THEN GOTO 650
630 IF i=12 THEN LET a$(4-e)="□":
    LET e=e+1: LET x=x-1
640 LET a$(4-e)=CHR$ i: PRINT #1;AT
    Ø,x;CHR$ i: LET x=x+1: LET e=e-1:
    IF e>Ø THEN PAUSE 1Ø: GOTO 590
650 IF e>1 AND (d=1 OR d=4 OR d=5)
    THEN GOTO 590
655 IF e>Ø AND (d=2 OR D=3) THEN
    GOTO 590
660 PAUSE 1Ø: PRINT #1;AT Ø,Ø;
    "□□□□□□□□□□□□□□□□
    □□□□□□□□□□□□□□□□
    □□":RETURN
670 LET i$=""
680 FOR z=1 TO 3
690 LET i$=i$+(a$(z) AND
    a$(z)<>"□")
700 NEXT z
710 IF LEN i$=3 THEN IF i$(1)<"A" OR
    i$(1)>"X" OR i$(2)<"Ø" OR
    i$(2)>"9" OR i$(3)<"Ø" OR
    i$(3)>"9" THEN LET f=1: RETURN
720 IF LEN i$=3 THEN IF VAL i$(2 TO 3)=Ø
    OR VAL i$(2 TO 3)>3Ø THEN LET f=1:
    RETURN
730 IF LEN i$=2 THEN IF i$(1)<"A" OR
    i$(1)>"X" OR i$(2)<"1" OR
    i$(2)>"9" THEN LET f=1: RETURN
740 IF d=2 THEN IF i$(1)<>"A" AND
    i$(1)<>"R" THEN LET f=1: RETURN
750 IF d=3 THEN IF i$(1)<>"C" AND
    i$(1)<>"R" THEN LET f=1: RETURN
760 LET z$(d,2 TO )=i$: LET z$(d,1)=CHR$
    (LEN i$+48): LET f=Ø: RETURN
770 LET fc=(CODE z$(4,2))-64: LET
    tc=(CODE z$(5,2))-64: LET fr=VAL
    z$(4,3 TO (1+VAL z$(4,1))): LET tr=VAL
    z$(5,3 TO (1+VAL z$(5,1)))
780 IF z$(3,2)="C" THEN IF fc<>tc OR
    fr>tr THEN LET f=1: RETURN
790 IF z$(3,2)="R" THEN IF fc>tc OR
    fr<>tr THEN LET f=1: RETURN
800 LET f=Ø: RETURN
810 FOR y=1 TO 3Ø: FOR x=1 TO 24: LET
    os=Ø
820 IF d$(y,x,17)="1" THEN LET z=CODE
    d$(y,x,18): GOSUB 880: GOSUB 1Ø1Ø:
    LET st=1: LET a$=STR$ t: LET os=LEN
    a$: IF t>99999.99 THEN LET
    d$(y,x,17)="5": RETURN
830 IF os>8 THEN LET st=os-7
840 IF os=Ø THEN GOTO 860
850 LET s$="□□□□□□□□": FOR
    u=st TO os: LET s$(u-st+1)=a$(u):
    NEXT u: GOSUB 141Ø: LET d$(y,x, TO
    8)=s$
860 LET i=IN 32766: IF i=252 THEN PRINT
    #1;AT Ø,Ø; PAPER 2; INK 7;"CALCULATE
    ABORTED": RETURN
870 NEXT x: NEXT y: RETURN
880 LET s$=d$(y,x,9 TO 16)
890 IF z=1 THEN LET v(1)=(CODE
    s$(1))-64: LET v(2)=VAL s$(2): LET
    v(3)=(CODE s$(3))-64: LET v(4)=VAL
    s$(4): LET o$=s$(5): RETURN
900 IF z=2 THEN LET v(1)=(CODE
    s$(1))-64: LET v(2)=VAL s$(2 TO 3):
    LET v(3)=(CODE s$(4))-64: LET
    v(4)=VAL s$(5): LET o$=s$(6): RETURN
```

```
860 A$=A$+"□"
870 A$=LEFT$(A$,LEN(A$)-1)
880 IF RIGHT$(A$,1)="□" THEN 870
890 PS=1:LN=LEN(A$):K=Ø
900 IF LN=1 AND A$="R" THEN F1=3:
    RETURN
910 IF LN=1 THEN F1=Ø:RETURN
920 IF LN=Ø AND RIGHT$(D$(R,C),8)=
    "□□□□□□□□" THEN A$=
    CHR$(128):F1=3:RETURN
930 GOSUB79Ø
940 PS=PS+1+D1
950 IF D1=Ø THEN F1=Ø:RETURN
960 IF PS<LN-1 THEN 930
970 IF PS>LN THEN F1=3:RETURN
980 Q1$=OP$:Q2$=MID$(A$,PS,1):
    GOSUB 267Ø
990 IF Q=Ø THEN F1=Ø:RETURN
1ØØØ D=VAL(RIGHT$(A$,1))
1Ø1Ø IF LN=PS THEN F1=3:RETURN
1Ø2Ø IF LN=PS+1 AND D>1 AND D<1Ø
     THEN F1=3:RETURN
1Ø3Ø F1=Ø:RETURN
1Ø4Ø IF A$=""THEN RETURN
1Ø5Ø GOSUB 86Ø
1Ø6Ø IF F1=Ø THEN A$="":INPUT
     "▢ENTER▮";A$:GOTO1Ø4Ø
1Ø7Ø RETURN
1Ø8Ø PRINT"▣"SPC(16)
     "▣WORKING▣"
1Ø9Ø FOR C=1 TO CM
11ØØ FOR R=1 TO RM
111Ø D=Ø:F=Ø:OP=8
112Ø F$=LEFT$(D$(R,C),8)
113Ø C1=ASC(MID$(F$,1,1))-64
114Ø IF C1=64 OR C1=-32 THEN 123Ø
115Ø IF C1=18 THEN GOSUB 132Ø:
     GOTO 123Ø
116Ø R1=VAL(MID$(F$,2,2)):IF R1>9 THEN
     F=1
117Ø C2=ASC(MID$(F$,3+F))-64:
     IF C2=-32 THEN GOSUB 136Ø:
     GOTO 123Ø
118Ø IF C2=-3 THEN D=VAL(MID$
     (F$,4+F,2)):GOSUB 136Ø:GOTO 123Ø
119Ø R2=VAL(MID$(F$,4+F,2)):
     IF R2>9 THEN F=F+1
12ØØ Q1$=OP$:Q2$=MID$(F$,5+F,1):
     GOSUB 267Ø
121Ø D=VAL(MID$(F$,6+F,2))
122Ø GOSUB 136Ø
123Ø NEXT R,C
124Ø RETURN
125Ø GET A$:IF A$="" THEN 125Ø
126Ø IF A$<AA$ OR A$>BB$ THEN 125Ø
127Ø RETURN
128Ø GET A$:IF A$="" THEN 128Ø
129Ø IF A$=AA$ THEN RETURN
13ØØ IF A$=BB$ THEN RETURN
131Ø GOTO 128Ø
132Ø AA$=RIGHT$(D$(R,C),8)
133Ø IF RIGHT$(AA$,1)="□" THEN
     AA$=LEFT$(F$,7):GOTO 133Ø
134Ø D$(R,C)=LEFT$(D$(R,C),8)+AA$
135Ø RETURN
136Ø V1=VAL(RIGHT$(D$(R1,C1),8)):V2=
```

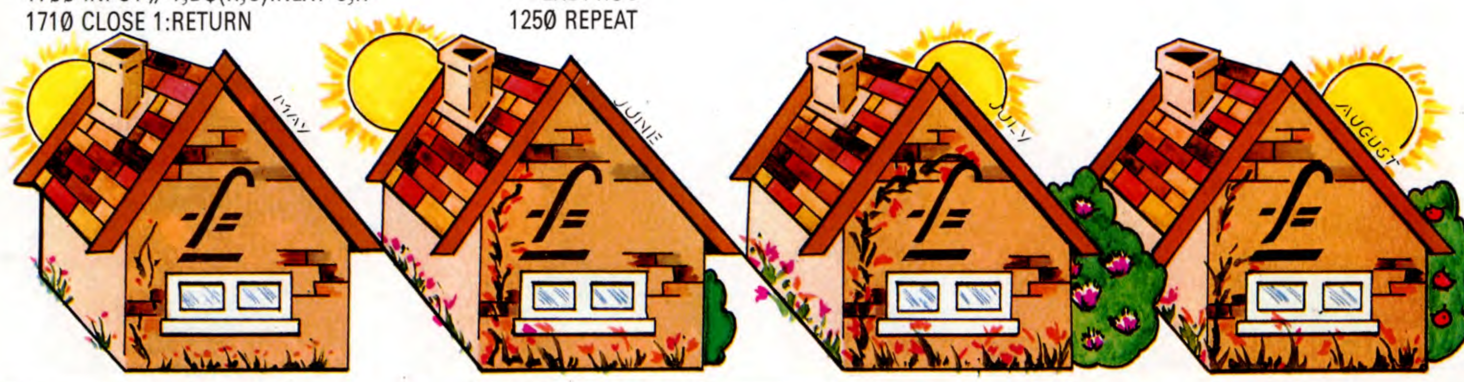


```
   VAL(RIGHT$(D$(R2,C2),8))
1370 ON Q GOSUB 1420,1430,1440,1450,
   1460,1470,1510,1550
1380 AA$=STR$(AN):IF LEN(AA$)>8
   THEN AA$="TOO□BIG"
1390 IF LEN(AA$)<8 THEN AA$="□"+
   AA$:GOTO 1390
1400 D$(R,C)=F$+AA$
1410 RETURN
1420 AN=V1+V2:RETURN
1430 AN=V1-V2:RETURN
1440 AN=V1*V2:RETURN
1450 AN=V1/V2:RETURN
1460 AN=V2*V1/100:RETURN
1470 AN=0:FOR NN=R1 TO R2
1480 AN=AN+VAL(MID$(D$
   (NN,C1),9,8))
1490 NEXT NN
1500 RETURN
1510 AN=0:FOR NN=C1 TO C2
1520 AN=AN+VAL(MID$(D$(R1,
   NN),9,8))
1530 NEXT NN
1540 RETURN
1550 AN=V1:RETURN
1560 PRINT"DO YOU WISH TO SAVE THIS
   DATA (Y/N)?"
1570 AA$="Y":BB$="N":
   GOSUB1280
1580 IF A$="N" THEN RETURN
1590 INPUT"■NAME";NM$:
   OPEN1,1,1,NM$
1600 PRINT#1,RM:PRINT#1,CM:
   FOR R=1 TO RM
1610 FOR C=1 TO CM
1620 PRINT#1,CHR$(34)+D$(R,C)+
   CHR$(34)
1630 NEXT C,R:CLOSE1:RETURN
1640 PRINT"■■■DO YOU WISH TO
   LOAD"
1650 PRINT"■AN EXISTING FILE (Y/N)?"
1660 AA$="Y":BB$="N":GOSUB1280
1670 IF A$="N" THEN RETURN
1680 INPUT"NAME";NM$:OPEN 1,1,
   0,NM$
1690 INPUT#1,RM,CM:FOR R=1 TO
   RM:FOR C=1 TO CM
1700 INPUT#1,D$(R,C):NEXT C,R
1710 CLOSE 1:RETURN
```

```
870 DEF FNformat
880 IF LENA$>8 A$=LEFT$(A$,8)
890 IF Type=8 AND LENA$<8 REPEAT
   A$=A$+"□":UNTIL LENA$=8
900 IF Type=0 AND LENA$<8 REPEAT
   A$="□"+A$:UNTIL LENA$=8
910 =A$
920 DEF FNboxcheck(pos)
930 LOCAL a$,b$,p,d
940 a$=MID$(A$,pos,3)
950 b$=LEFT$(a$,1)
960 IF b$<"A" OR b$>"X" =0
970 p=VAL(RIGHT$(a$,2))
980 d=2:IF p<10 d=1
990 IF p>Cols OR p<1 =0
```

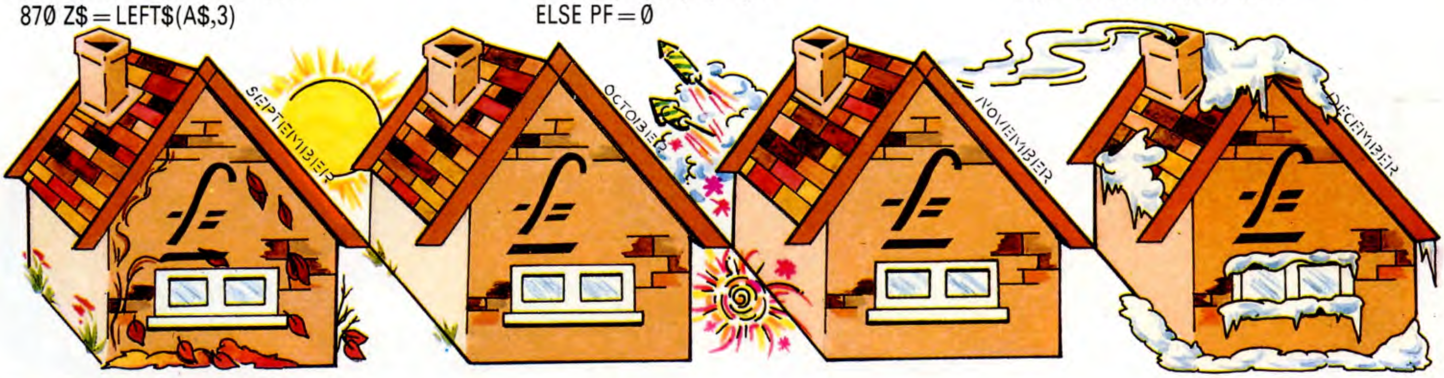
N.B. In Line 990, Electron users should change Cols to ROWS

```
1000 =d
1010 DEF FNCheck
1020 LOCALpos,ln,f,d,k
1030 A$=A$+"□"
1040 REPEAT A$=LEFT$(A$,LENA$-1)
1050 UNTIL RIGHT$(A$,1)<>"□"
1060 pos=1:ln=LENA$
1070 IF ln=1 AND A$="R" =3
1080 IF ln=1 =0
1090 IF ln=0 AND RIGHT$(D$(Row,Col),
   8)=STRING$(8,"□") A$=CHR$(128):
   =3
1100 REPEAT
1110 k=k+1
1120 f=FNboxcheck(pos)
1130 pos=pos+1+f
1140 UNTIL f=0 OR pos>=ln OR k=2
1150 IF f=0 =0
1160 IF pos>ln =3
1170 IF INSTR(Op$,MID$(A$,pos,1))=0 =0
1180 d=VAL(RIGHT$(A$,1))
1190 IF ln=pos =3
1200 IF ln=pos+1 AND d>-1 AND
   d<10 =3
1210 =0
1220 DEF PROCformulacheck
1230 LOCALvpos,f
1240 vpos=VPOS+1:IF A$="" THEN
   ENDPROC
1250 REPEAT
1260 f=FNCheck
1270 IF f=0 PRINTTAB(0,vpos)CHR$129;
   "INCORRECT:PLEASE RE-DO□";:INPUT
   ""A$
1280 UNTIL f>0
1290 ENDPROC
1300 DEF PROCcalculate
1310 PRINTCHR$129;CHR$136;"WORKING"
1320 FOR c%=1 TO Cols
1330 FOR r%=1 TO Rows
1340 d%=0: f%=0:op%=8
1350 F$=LEFT$(D$(r%,c%),8)
1360 c1%=FNc(1)
1370 IF c1%=64 OR c1%=-32 GOTO 1470
1400 r1%=FNr(2):IF r1%>9 f%=1
1410 c2%=FNc(3+f%):IF c2%=-32
   PROCarith:GOTO1470
1420 IF c2%=-3 d%=FNr(4+f%):
   PROCarith:GOTO1470
1430 r2%=FNr(4+f%):IF r2%>9
   f%=f%+1
1440 op%=INSTR(Op$,MID$(F$,5+f%,1))
1450 d%=FNr(6+f%)
1460 IF op%<>0 THEN PROCarith
1470 NEXT
1480 NEXT
1490 ENDPROC
1500 DEF FNr(p%)=VAL(MID$
   (F$,p%,2))
1510 DEF FNc(p%)=ASC(MID$
   (F$,p%,1))-64
1520 DEF FNv1=VAL(RIGHT$(D$(r1%,
   c1%),8))
1530 DEF FNv2=VAL(RIGHT$(D$(r2%,
   c2%),8))
1540 DEF PROCgetbet(a$,b$)
1550 REPEAT A$=GET$
1560 UNTIL A$>a$ AND A$<b$
1570 ENDPROC
1580 DEF PROCgetabs(a$,b$)
```



```
1590 REPEAT A$=GET$:UNTIL A$=a$ OR
   A$=b$
1600 ENDPROC
1670 DEF PROCarith
1680 LOCALans,a$,d
1690 ON op% GOSUB 1780,1790,1800,1810,
   1820,1830,1870,1910
1700 IF d%=Ø THEN ans=INT(ans+.5):
   GOTO 1720
1710 @%=&1020008+d%*256
1720 a$=STR$(ans):IF LENa$>8
   a$=CHR$135+"TOO BIG"
1730 IF LENa$<8 a$="□"+a$:
   GOTO1730
1740 D$(r%,c%)=F$+a$
1750 @%=&90A
1760 ENDPROC
```

```
660 A$=D$(I,J):B$=MID$(A$,2)
670 AT=ASC(A$)
680 IF AT=128 THEN PRINTSTRING$(7,32);:
   GOTO 720
690 IF AT=129 OR AT=130 THEN PRINT
   USING "%□□□□□%";B$;:
   GOTO 720
700 FOR U=1 TO LEN(B$):IF MID$
   (B$,U,1)<>CHR$(32) THEN PRINT
   MID$(B$,U,1);
710 NEXTU
720 RETURN
730 C1=ASC(Z$)-64:C2=VAL(MID$
   (Z$,2)):V=D(C1,C2):RETURN
740 PRINT@448,"WORKING"
750 FOR J=1 TO RX
760 FOR I=1 TO CX
770 D(I,J)=Ø:IF ASC(D$(I,J))=129 THEN
   D(I,J)=VAL(MID$(D$(I,J),2))
780 NEXT I,J
790 FOR J=1 TO RX
800 FOR I=1 TO CX
810 PRINT@448,"WORKING ON CELL□";
   CHR$(I+64);MID$(STR$(J),2)
820 IF ASC(D$(I,J))<>131 THEN 1130
830 A$=MID$(D$(I,J),2)
840 O$=MID$(A$,7,1)
850 IF O$="&" THEN 1050
860 IF O$="$" THEN 1090
870 Z$=LEFT$(A$,3)
880 GOSUB 730
890 V1=V:Z$=MID$(A$,4,3):GOSUB 730:
   V2=V
900 DP=VAL(RIGHT$(A$,1))
910 ON INSTR(1,OP$,O$) GOSUB
   1000,1010,1020,1030,1040
920 OV=Ø:IF DP=Ø THEN PU$=
   "#######":MP=7:
   GOTO 950
930 PU$=STRING$(7-(DP+1),"#")+
   "."+STRING$(DP,"#"):MP=7-
   (DP+1)
940 IF LEN(PU$)>7 THEN RV$=
   "□□<OV>":OV=1
950 D(I,J)=RV
960 IF RV<Ø THEN MP=MP-1
970 ML=LEN(MID$(STR$(INT(RV+.5)),2))
980 IF ML>MP THEN RV$="□□
   <OV>":OV=1
990 GOTO 1160
1000 RV=V1+V2:RETURN
1010 RV=V1-V2:RETURN
1020 RV=V1*V2:RETURN
1030 IF V2=Ø THEN RV=Ø:RETURN ELSE
   RV=V1/V2:RETURN
1040 RV=V1*V2/100:RETURN
1050 P1=ASC(A$)-64:P2=ASC(MID$
   (A$,4,1))-64:C2=VAL(MID$(A$,2,2)):
   RV=Ø
1060 FOR C1=P1 TO P2
1070 RV=RV+D(C1,C2):NEXT
1080 DP=VAL(RIGHT$(A$,1)):
   GOTO 920
1090 P1=VAL(MID$(A$,2,2)):P2=VAL
   (MID$(A$,5,2)):C1=ASC(A$)
   -64:RV=Ø
1100 FOR C2=P1 TO P2
1110 RV=RV+D(C1,C2):NEXT
1120 DP=VAL(RIGHT$(A$,1)):
   GOTO 920
1130 IF ASC(D$(I,J))<>128
   THEN 1150
1140 RV$=STRING$(7,32):GOTO 1160
1150 RV$=MID$(D$(I,J),2)
1160 IF I>=CS AND I<=CS+3 AND
   J>=RS AND J<=RS+11 THEN
   PRINT@(J-RS)*32
   +35+(I-CS)*7,"";:PF=1
   ELSE PF=Ø
1170 IF (ASC(D$(I,J)) >= 128 AND
   ASC(D$(I,J)) <=130) OR OV=1 THEN
   1200
1180 IF PF=1 THEN PRINT USING
   PU$;RV;
1190 GOTO 1210
1200 IF PF=1 THEN PRINTUSING
   "%□□□□□%";RV$;
1210 NEXT I,J
1220 RETURN
1230 CLS:INPUT "DO YOU WANT TO SAVE
   THIS SHEET□□(Y/N)";A$
1240 IF A$<>"Y" THEN 1340
1250 LINE INPUT "FILENAME:";F$
1260 OPEN "O",#-1,F$
1270 FOR J=1 TO RX
1280 FOR I=1 TO CX
1290 IF ASC(D$(I,J))=128 THEN 1320
1300 Z$=D$(I,J):MID$(Z$,1,1)=CHR$
   (ASC(MID$(Z$,1,1))-95)
1310 PRINT#-1,STR$(I),STR$(J):
   PRINT#-1,Z$
1320 NEXTI,J
1330 CLOSE#-1
1340 CLS:MO=1:GOSUB 70:RETURN
1350 CLS:INPUT "DO YOU WANT TO LOAD A
   SHEET FROMTAPE (NOTE SHEET IN
   MEMORY WILL BE MERGED WITH THAT
   ON TAPE)□□□□(Y/N)";A$
1360 IF A$<>"Y" THEN 1480
1370 PRINT"PRESS enter TO LOAD NEXT FILE
   ONTAPE- OR INPUT FILENAME
   NOW":PRINT
1380 LINE INPUT "FILENAME:";F$
1390 OPEN "I",#-1,F$
1400 IF EOF(-1) THEN 1470
1410 INPUT#-1,A$,B$:LINE
   INPUT#-1,C$
1420 MID$(C$,1,1)=CHR$(ASC(MID$
   (C$,1,1))+95)
1430 C1=VAL(A$):C2=VAL(B$):D$(C1,
   C2)=C$
1440 IF C1>CX THEN CX=C1
1450 IF C2>RX THEN RX=C2
1460 GOTO 1400
1470 CLOSE#-1
1480 CLS:CC=1:CR=1:CS=1:RS=1:
   MO=1:GOSUB 70:RETURN
```



CUMULATIVE INDEX

An interim index will be published each week. There will be a complete index in the last issue of *INPUT*.

A

Animation
of UDGs in cliffhanger 992–997
using colour fill techniques
Acorn 955–959
using GCOL 3
Acorn 999–1000
using paged graphics
1022–1027, 1132–1137

Applications
calendar and diary program
1010–1016, 1017–1021, 1064–1067
hobbies file, extra options 947–952
magnification program 1081–1087
spreadsheet program
1118–1126, 1172–1176
text-editor program
852–856, 878–883, 914–920

B

BASIC
adding instructions to
Acorn, Dragon, Spectrum 844–851

Basic programming
analyzing and storing sounds 1091–1095
animation with paged graphics
1022–1027, 1132–1137
colour commands, *Acorn* 953–959
Computer Aided Design 998–1004
designing a new typeface 838–843
drawing conic sections 859–863, 889–895
how programs are stored 1106–1112
mathematics of growth 1049–1056
mechanics, principles of 933–939
multi-key control 974–979
musical chords and harmonies 985–991
patterns from nature 1164–1171
prediction by computer 1158–1163
programming function keys 825–829
secret codes 960–965, 1044–1048
sound envelopes
Acorn, Commodore 64 1138–1144
speeding up BASIC programs 921–927

C

Calendar and diary program
1010–1016, 1017–1021, 1064–1067

Chords, musical
definition 985–986
programs to play
Acorn, Commodore 64 986–991

Cliffhanger game
part 1—title page 904–913
part 2—adding instructions 928–932
part 3—adding a tune 966–973
part 4—graphics and merging 992–997
part 5—setting the scene 1034–1043
part 6—perils and rewards 1057–1063
part 7—initializing routine 1101–1105
part 8—synchronizing routine
1127–1131
part 9—scoring routine 1145–1151

Codes, secret 960–965, 1044–1048

Colour
defining in machine code 1034–1043
filling in with
Acorn 953–959
in Teletext mode
BBC 1068–1073
routines for changing
Commodore 64 872–877

Computer Aided Design
rubber-banding and picking
and dragging 998–1004

Conic sections 857–863, 889–895

D

Digital clock routine 896–898
Dynamics, programs to
illustrate 1164–1171

E

Envelope,
of orbits 1164–1167
sound
Acorn, Commodore 64
968–971, 1138–1144
in musical harmony programs 986–991

F

Filling in with colour
Acorn 953–959

Fish population program 1170–1171

Fox and geese game
part 1—principles and graphics
1096–1100
part 2—initializing and
mapping the moves 1113–1117
part 3—higher levels 1152–1157

Fruit machine game
1028–1033, 1074–1080

Function keys, programming
Acorn, Commodore 64, Vic 20 826–829

G

Games
cliffhanger 904–913, 928–932, 966–973,
992–997, 1034–1043, 1057–1063, 1101–1105,
1127–1131, 1145–1151
fox and geese
1096–1100, 1113–1117, 1152–1157
fruit machine 1028–1033, 1074–1080
goldmine 830–837, 864–871
lunar touchdown 1088–1090
magnification 1081–1087
multi-key control for 974–979
othello 980–984, 1005–1009
wordgame 899–903, 940–945

Goldmine game 830–837, 864–871

Graphics
colour commands, *Acorn* 953–959
effects using curves 857–863, 889–895
hi-res
for custom typeface 838–843
setting up new commands
Commodore 64 872–877
magnification program for 1081–1087
paged, for animation
1022–1027, 1132–1137
patterns from nature 1164–1171
picking and dragging 1000–1004
rubber-banding 998–1000
trace of sound 1092–1095
using Teletext mode, *BBC* 1068–1073

Growth, measuring 1049–1056

H

Hobbies file, extra options for 947–952

I

Instructions, adding to BASIC
Acorn, Dragon, Spectrum 844–851

K

Keypresses, multiple, programming for
974–979

L

Letter-generator program 838–843
Lunar touchdown game 1088–1090

M

Machine code
games programming
see cliffhanger
merging routines 992–997
routines for hi-res graphics
Commodore 64 872–877
routine to alter BASIC 844–849
timer routine 896–898
tune routine 966–973

Magnification program 1081–1087

Mathematical functions
in mechanics 935
in spreadsheet program 1120
speedy use of 923–924
to assess population tendencies
1170–1171
to draw curves 857–863, 889–895
to draw patterns from orbits
1164–1170
to measure growth 1049–1056

Mechanics
programs to show principles of 933–939

Memory
how BASIC programs are stored in
1106–1112
paged graphics in 1023–1027, 1132–1137

Multi-key control, programming for
974–979

Music
analyzing and storing 1091–1095
chords and harmonies 985–991
machine code routine for 966–973

N

Numbers
Fibonacci 1056
generation program 1054–1056

O

Orbits, patterns from 1164–1171
Othello board game
980–984, 1005–1009

P

Paged graphics 1023–1027, 1132–1137
Patterns from nature 1164–1171
PLOT
new commands, *Acorn* 953–959
Pools simulation program 1158–1160
Prediction by computer 1158–1163

R

RND function
in computing probability 1160–1163
Robotics 884–888

S

Search routines
binary and serial 924–927
in text-editor program 914–920
single pass 1162–1163

Sounds
analyzing and storing 1091–1095
envelopes for modifying
Acorn, Commodore 64 1138–1144

Sort routines
in hobbies file program 947–953
in text-editor program 914–920

Speeding up BASIC programs 921–927

Spreadsheet program
part 1 1118–1126
part 2 1172–1176

T

Teletext mode, *BBC* 1068–1073

Text-editor program
part 1—basic routines 852–856
part 2—editing facilities 878–883
part 3—sorting, searching,
formatting and printout 914–920

Timer routine
for BASIC lines 922
machine code 896–898

Typeface. setting up new 838–843

V

Variables
managing for program speed 923–925
setting in machine code game 1127–1131
storing in memory 1106–1112

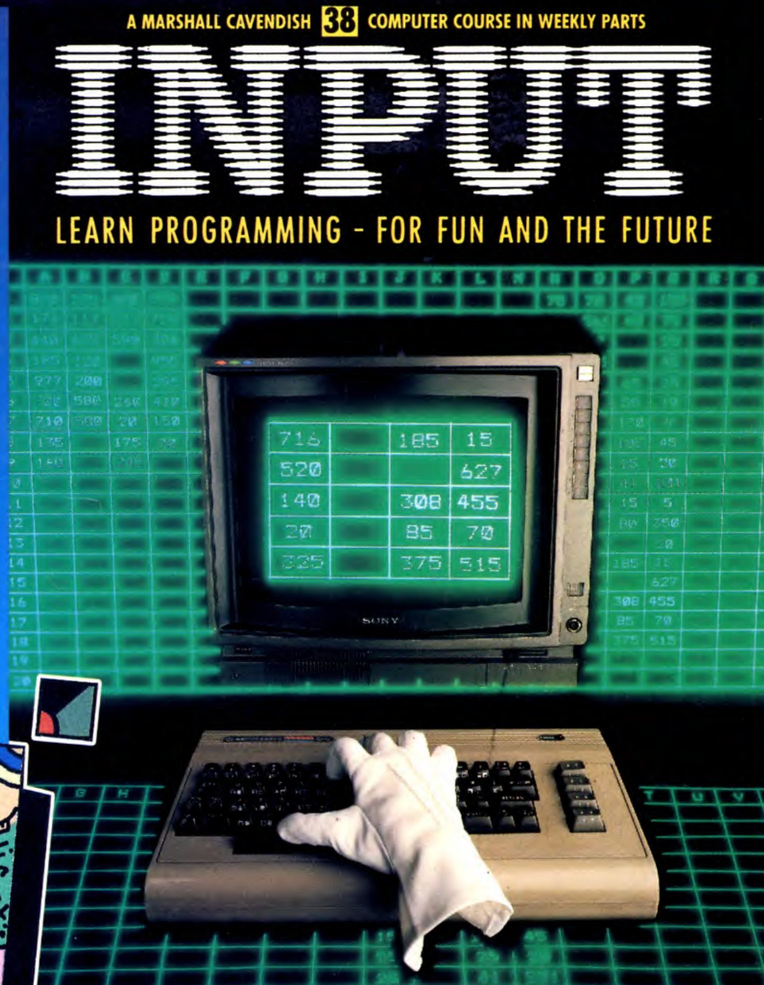
W

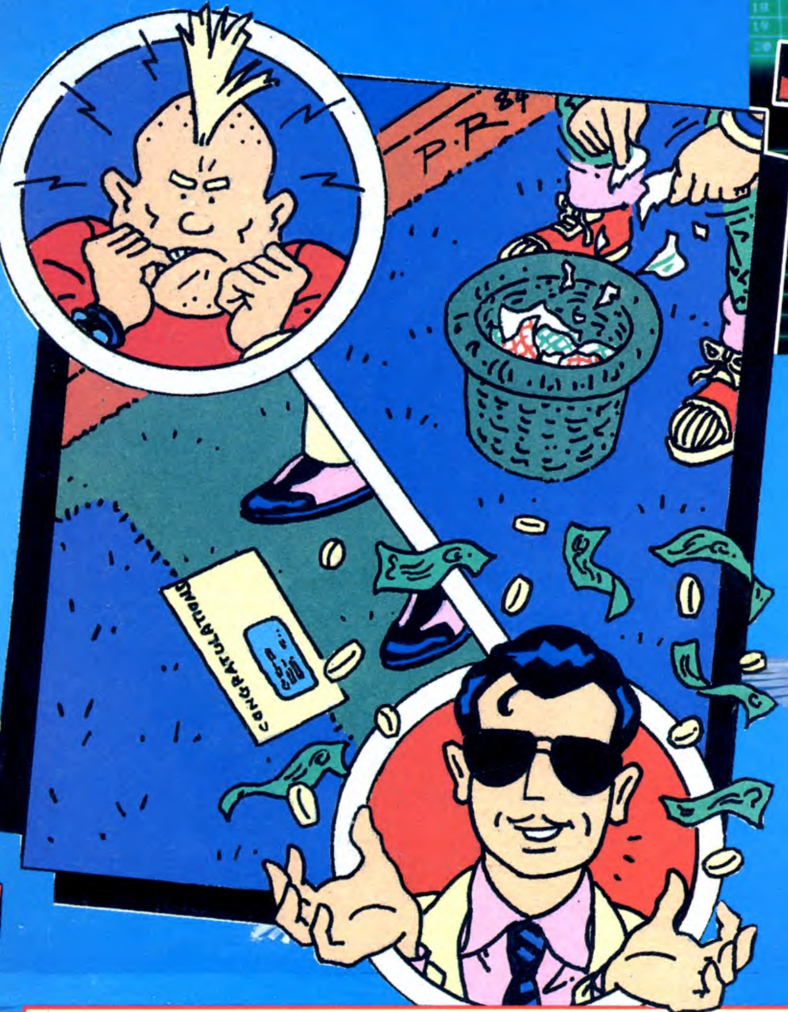
Waveforms
displaying and storing 1092–1095
modulation of 1138–1139, 1142

Wordgame 899–903, 940–945

COMING IN ISSUE 38...

- *A fearsome Martian spider, a neurotic window cleaner, a bevy of balloons and an accumulation of arrows are all part of FREDDY AND THE SPIDERS FROM MARS, our BASIC arcade game*
- *Complete the SPREADSHEET program and plan your finances*
- *Got ten thumbs? Can't draw a thing? GRAPHICS OF ROTATION allows you to create realistic three-dimensional views of any shape you choose*
- *Use statistical techniques to MODEL REALITY. Find out how to apply statistical models*
- *Sight some circling seagulls in CLIFFHANGER on the SPECTRUM, COMMODORE and ACORN machines*







ASK YOUR NEWSAGENT FOR INPUT